
64'er
SONDERHEFT
BASIC
SONDERHEFT 40
ÖS 100,–/Sfr. 14,–
Lit. 14000/hfl. 18,–/dkr. 72,–
DM 14,–
Markt&Technik
64'er
Der Kurs
BASIC
Schritt für Schritt
Grundlagen
Keine Chance
für Fehler
■ Effektive Fehlersuche
in Programmen
■ So werden
Ihre Programme schneller
■ Der richtige Umgang
mit der Floppy
Listings
Profi-Tools
und viele Tips
■ ExBasic Level II:
Das Kraftpaket – über
70 neue Befehle
■ Basic-Kontroll-System:
Der Detektiv
für Programmierfehler
■ Einzeiler:
Miniprogramme mit Pfiff
IF
FOR
BASIC
PRINT
NEXT
Alle Programme auch auf Diskette erhältlich



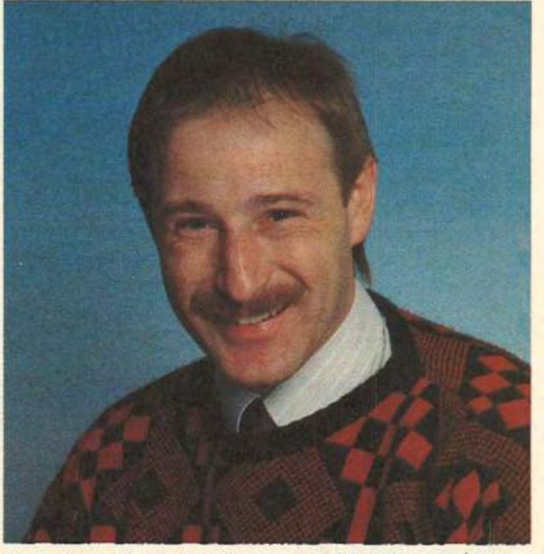
EDITORIAL

Basic macht Geschichte

● Wie die Menschen kennt auch der Computer eine Vielzahl von Sprachen. Pascal, Fortran oder Cobol sind nur einige davon. Die meisten dieser Computersprachen sind auf bestimmte Verwendungen zugeschnitten. Cobol beispielsweise ist besonders für kaufmännische Anwendungen geeignet, während die Stärke von Fortran auf dem mathematischen Bereich liegt.

● Wenn Sie Ihren C64 einschalten, können Sie sich mit ihm in »Basic« unterhalten. Die Programmiersprache »Basic« wurde bereits 1963 entwickelt und ist eine Abkürzung aus dem Englischen. Voll ausgeschrieben bedeutet es: Beginners All Purpose Symbolic Instruction Code (auf deutsch etwa: Anfänger-Sprache für jeglichen Einsatz). Bei der Entwicklung von Basic achtete man darauf, daß die Sprache einfach zu lernen und universell einsetzbar ist.

● Mehr als 1,5 Millionen verkaufte C64 haben nicht unerheblich zur Verbreitung von Basic beigetragen. Dennoch muß man sagen, daß das Handbuch, welches jedem C64 beim Kauf beiliegt, eher als »magerer Einstieg« denn gelungene Literatur für Basic zu bezeichnen ist.



● Wir lassen Sie mit dem Handbuch nicht allein. Ein ausführlicher Basic-Kurs nimmt Sie quasi an die Hand und geleitet Sie – ohne Vorkenntnisse vorauszusetzen – in das Land der Basic-Programmierer. Wenn Sie den Kurs »Schritt für Schritt« mitgemacht haben, dürfen Sie sich durchaus als fortgeschrittener Programmierer bezeichnen.

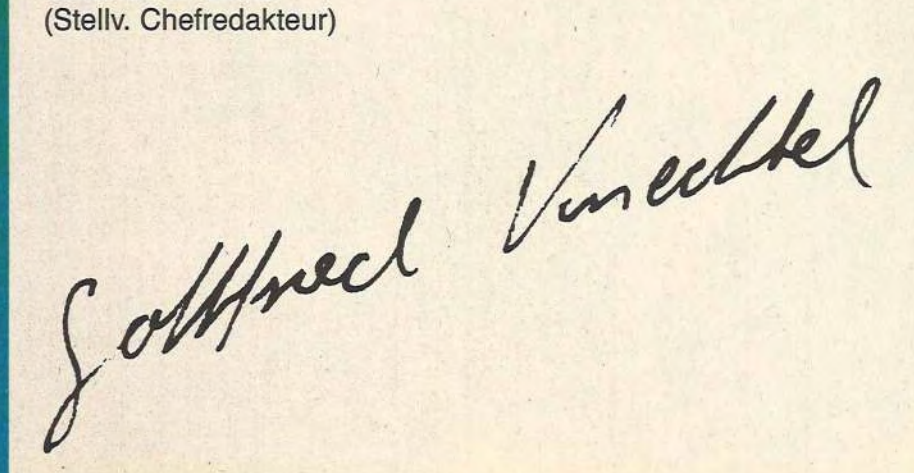
● So einfach, wie das Basic 2.0 gehalten ist, läßt es natürlich auch Wünsche nach zusätzlichen Befehlen offen. Wir zeigen Ihnen, welche wichtigen Basic-Erweiterungen es gibt und wo ihre Stärken liegen. Eine früher kommerziell vertriebene Basic-Erweiterung können wir Ihnen sogar zum Abtippen bieten. »ExBasic Level II« wartet mit mehr als 70 neuen Befehlen auf.

● Neben dem umfangreichen Kurs finden Sie noch weitere wichtige Grundlagen. Beim Programmieren unterläuft einem schnell ein Fehler. Auch für diese Fälle überlassen wir Sie nicht Ihrer Verzweiflung. Sie erfahren, wie man Fehler systematisch suchen kann.

Jede Menge Tips, Tricks und hilfreiche Programme runden das Heft für alle Basic-Freunde ab.



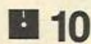
Ihr
Gottfried Knechtel
(Stellv. Chefredakteur)



INHALT

Übersicht

Listings



DTP



Kurs

Gewinnen Sie bei unserem Wettbewerb: Bücher, die für jeden C64-Besitzer unentbehrlich sind. Seite 140

Grundlagen

Ein Basic-Kurs der Superlative. Schritt für Schritt werden Sie mit dieser Programmiersprache vertraut gemacht. So wird auch ein völliger Laie zum fortgeschrittenen Programmierer. Seite 48

BASIC FÜ
Schritt für Schritt.

»ExBasic Level II«, diese professionelle Basic-Erweiterung gibt es in dieser Ausgabe zum Abtippen. Über 70 neue Befehle bieten riesigen Komfort. Seite 10





Warum gleich Geld für ein teures Keyboard ausgeben? Das Programm »Synthesizer« verwandelt Ihren C64 in einen solchen. Experimentieren Sie doch mit dem Computerinstrument.
Seite 45

Tips & Tricks

Eingabehilfen

Sonstiges

Alle Programme aus Artikeln mit einem Disketten-Symbol finden Sie auch auf der Programmservice-Diskette zu diesem Sonderheft

Warum sich die Augen ruinieren bei der Fehlersuche in Basic-Programmen? Es geht viel leichter: Das Basic-Kontroll-System erkennt für Sie Syntax-Fehler und sogar unsauberen Programmierstil.
Seite 24

Knobelecke



Mit dem Basic 2.0 des C 64 können Sie schon eine Menge anfangen. Aber Gutes läßt sich noch verbessern. Hier einige der wichtigsten Basic-Erweiterungen: Machen Sie mehr aus Ihrem Computer.

Wer sich intensiver mit Basic-Programmierung beschäftigt, wird sich nach einiger Zeit zusätzliche Befehle wünschen, die beispielsweise das Editieren erleichtern. Im Handbuch erfahren Sie, wie mit dem vorhandenen Befehlssatz des C 64 Sounds, Sprites oder Grafik programmiert werden können. Es geht aber auch einfacher, wie der C 128 mit seinem Basic 7.0 zeigt.

Sehr nützlich ist zum Beispiel ein AUTO-Befehl: Bei längeren Programmen, die Sie selbst schreiben oder abtippen, ist es angenehm, wenn nach einem <RETURN> zum Abschluß einer Zeile nicht immer wieder die neue Zeilennummer eingetippt werden muß.

Programmierer erleben häufig folgendes Problem: Die Zeilennumerierung wird in Zehner-Schritten gewählt, um nachträgliche Ergänzungen einbauen zu können. Bei vielen Änderungen reicht dann der Platz nicht mehr, weil keine freien Zeilennummern vorhanden sind. Also müßte das Programm umnumeriert werden. Aber wie? Alle Zeilennummern wieder neu schreiben? Wenn innerhalb von Zeilen die Sprungziele eines <GOTO> oder <GOSUB> nicht zusätzlich geändert werden, kann es zu merkwürdigen Überraschungen im Programmablauf kommen. Abgesehen davon, daß dieser Weg bei längeren Programmen ohne Fehler kaum durchführbar ist: Es fehlt der entspre-

chende RENUMBER-Befehl, mit dem die Umnumerierung sehr einfach ausgeführt werden kann.

Haben Sie – nach einer langen Programmiersitzung – unabsichtlich NEW eingegeben? Wer vorher vergessen hat, das Ergebnis der stundenlangen Arbeit auf Diskette oder Datasette zu speichern, wird nicht besonders erfreut sein: Das Programm ist verschwunden. Zwar gibt es Tricks, mit denen das Programm gerettet werden kann, aber nicht jeder weiß, wie er dazu vorgehen soll. Wohl dem, der eine Basic-Erweiterung mit dem rettenden OLD-Befehl besitzt.

Die Darstellung von Sprites ist eine fantastische Eigenschaft des C 64. Eine professionelle Programmierung von Sprites in Basic erfordert aber gute Kenntnisse des VIC (Video Interface Chip) und einen intensiven Gebrauch des POKE-Befehls.

Nicht viel anders sieht es mit der Grafikprogrammierung aus. Haben Sie schon einmal versucht, einen Kreis auf dem Bildschirm darzustellen? Statt eines einfachen CIRCLE-Befehls gehören beim C 64 einige Vorbereitungen und Berechnungen dazu.

Die genannten Schwierigkeiten führten schnell zur Veröffentlichung von Programmen, die den Befehlssatz des Basic ein oder zwei nützliche Zusatzbefehle integrierten. Ebenso wurden spezielle Erweiterungen vorgestellt, die Spezialisten auf Gebieten wie Sound- oder Grafikprogrammierung waren.



Eine der ersten Befehlserweiterungen, die sehr umfangreiche Ergänzungen und Neuerungen bot, war Simons Basic. Ein wesentlicher Vorteil dieser Basic-Erweiterung war, daß sie als Modul vertrieben wurde: Nach dem Einstecken des Moduls in den Expansions-Port des C 64 verfügte man sofort über eine Vielzahl von neuen Befehlen.

Simons Basic brachte neben Programmierhilfen wie <AUTO> oder <DUMP> Befehle, die strukturiertes Programmieren erlaubten (beispielsweise LOOP...END LOOP, IF...THEN...ELSE). Während das Einschalten einer hochauflösenden Grafik im Standard-Basic einige Zeilen mit vielen POKEs füllte, genügte in Simons Basic der HIRES-Befehl.

Simons Basic war für viele Programmierer eine wahre Fundgrube: Es gab kaum einen Bereich, für den es keinen sinnvollen Befehl gab. Ob Joystickabfragen oder Abfragen für Light-Pen und Paddle, ob komfortable Spriteprogrammierung und eine Bildschirmsteuerung: Mit Simons Basic wurde die Basic-Programmierung erheblich vereinfacht.

Simons Basic wird zwar heute nicht mehr vertrieben, ist aber bei vielen C 64-Besitzern noch zu finden, da es es einige Zeit auch auf Diskette von Commodore angeangeboten wurde. Wer Programme für private Zwecke schreiben will, findet mit Simons Basic eine hervorragende Unterstützung.

Lange Zeit wurde Exbasic Level II nur kommerziell vertrieben: In diesem Sonderheft finden Sie das komplette Listing zu Exbasic Level II zum Abtippen.

Über 70 neue Befehle erleichtern die Basic-Programmierung. Eine Aufstellung der neuen Befehle mit kurzen Erläuterungen und Beispielen finden Sie ab Seite 10. Wer sich nicht die Mühe des Abtippens machen will, hat zwei Möglichkeiten: Auf der Programmservice-Diskette zu diesem Sonderheft ist neben allen anderen Listings auch Exbasic Level II enthalten. Sie können diese Basic-Erweiterung auch als Modul bei der im Info am Ende dieses Artikels angegebenen Adresse bestellen. Dieses Modul des C 64 wird in den Expansions-Port hinten rechts am Gehäuse eingesteckt.

Ein Modul, das neben einer umfangreichen Basic-Erweiterung eine Vielzahl sinnvoller Funktionen enthält (Bild 1), ist Magic Formel. Wir beschränken uns hier im Rahmen dieses Artikels auf die zusätzlichen Befehle.

Bereits die Einschaltmeldung mit »116kROM, 72kRAM, 38911 BASIC Bytes free« deutet die Leistungsfähigkeit der Erweiterung an. Die neuen Basic-Befehle können in drei Gruppen aufgeteilt werden: »Toolkit«, »DOS-Befehle« und »Grafik«.

Ein wichtiger Befehl ist <HELP>: Er listet alle zur Verfügung stehenden Basic-Befehle auf dem Bildschirm auf.

Zu den Toolkit-Befehlen gehören beispielsweise eine automatische Zeilennumerierung, ein Renumber-Befehl, oder das Retten von gelöschten Basic-Programmen mit <OLD>. Sehr hilfreich ist der Suchbefehl <FIND>, der in einem Basic-Programm nach einer von Ihnen anzugebenden Zeichenkette sucht. Wer ein Listing auf dem Drucker ausgeben will, kann dies sehr einfach mit <LLIST> erledigen; dieser Befehl ersetzt die im Standard-Basic übliche Sequenz »OPEN 4,4:CMD 4:LIST:CLOSE 4«.

Zu den DOS-Befehlen, also den Befehlen, die sich mit dem Diskettenbetrieb befassen, gehören beispielsweise <DLOAD> und <DSAVE>, die ein Basic-Programm wesentlich schneller laden und speichern als unter dem Standard-Basic.

Besitzer eines zweiten Diskettenlaufwerks werden den Befehl <CDRIVE> begrüßen, mit dem ein Laufwerk sehr schnell auf Geräteadresse 9 umgestellt werden kann.

Wollen Sie ein Basic-Programm unter Magic Formel laden und starten, das nicht mit dem Modul zusammenarbeitet, verwenden Sie den Befehl <CRUN>: Der Rechner wird vor dem Starten des Programms in den Originalzustand zurückgesetzt. Sie kommen aber immerhin noch in den Genuß der Schnelladeroutine.



Bild 1. Mit der Funktionstaste <F5> aktivieren Sie im Hauptmenü die Basic-Erweiterung

Die Grafikbefehle unterstützen hochauflösende Grafik mit 320 x 200 Punkten und Multicolor-Grafik mit 160 x 200 Punkten. Bis zu vier Grafikseiten werden verwaltet.

Mit den vorhandenen Befehlen fällt es nicht schwer, ansprechende Grafiken zu erstellen: Jede Grafikseite kann gezielt angewählt werden, Sie können Punkte setzen ((PLOT)), Linien zeichnen (<LINE> oder <DRAW>), Rechtecke (<BOX>) oder Kreise (<CIRCLE>) erstellen. Flächen können mit verschiedenen Farben ausgefüllt werden, eine gesamte Grafik invertiert werden.

Magic Formel ist keine spezielle Basic-Erweiterung, bietet es doch eine Menge zusätzlicher Funktionen. Der vorhandene Befehlssatz erleichtert das Editieren von Programmen, vereinfacht die Arbeit mit dem Laufwerk und unterstützt das Programmieren von Grafiken. Basic-Befehle von Magic Formel finden Sie in Tabelle 1 aufgelistet.

Mit dem Hyper-Basic-Modul (Bild 2) wird ein effektives Programmieren in Basic erheblich vereinfacht: Über 100 neue Basic-Befehle unterstützen String- und Soundbearbeitung ebenso wie Grafik- oder Sprite-Programmierung.

assembler	auto	cdrive	cat
dappend	delete	dec	dir
dload	dsave	dverify	config
find	help	hex	jump
llist	lprint	off	old
renum	crun	send	status
hires	multi	clear	plot
invert	line	text	graphik
page	box	draw	mix
copy	circle	gsave	gload
frame	hprint	vprint	block
fill	replace	lrun	

Tabelle 1. Basic-Befehle von Magic Formel im Überblick

Das integrierte Hardcopy-Modul läßt kaum noch Wünsche offen, vor dem Ausdruck einer Grafik können Sie diese beispielsweise noch editieren. Die Befehle zur Arbeit mit dem Diskettenlaufwerk kennen Sie von der Utility auf der Test/Demo-Diskette, Ladevorgänge werden zirka um den Faktor 12 beschleunigt.

Da Hyper-Basic eine Fülle von Befehlen bietet, wollen wir Ihnen nur einige ausgesuchte Leckerbissen vorstellen. Die Liste der Befehle finden Sie in Tabelle 2.

Eine komplette Befehlsübersicht auf dem Bildschirm können Sie sich durch Eingabe von <HELP2> oder durch Drücken der Tasten <SHIFT F1> ausgeben lassen. Vorbildlich ist bei der Übersicht, daß nicht nur die Befehle, sondern auch die Syntax dargestellt werden. Das erspart oft den Blick ins (ausführliche) Handbuch.

Die Funktionstasten sind nach dem Einschalten mit häufig verwendeten Befehlen belegt. Sie können aber jederzeit mit <KEY> eine eigene Belegung vornehmen. Durch Einbeziehung der CTRL- und CBM-Taste stehen Ihnen insgesamt sechzehn Funktionstasten zur Verfügung.

Als Basic-Spezialist bietet das Modul Befehle zur strukturierten Programmierung: Mit <DO WHILE>, <LOOP UNTIL> oder <EXIT IF> lassen sich einfache Schleifen konstruieren, aus denen man an beliebiger Stelle aussteigen kann.

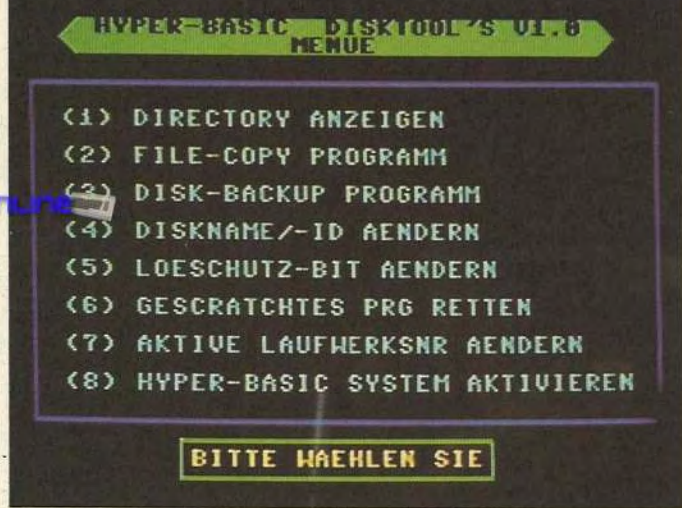


Bild 2. Über 100 neue Basic-Befehle plus Disketten-Utility: Das Hyper-Basic-Modul läßt kaum noch Wünsche offen.

Das Abfangen von Fehlern innerhalb eines Programms erfolgt mit <ERRINIT>. Tritt im Verlauf des Programms ein Fehler auf, wird keine Fehlermeldung ausgegeben. Geben Sie hinter dem Befehl zum Beispiel ein <GOTO 100> ein, wird zu einer Fehlerbehandlungs-Routine ab Zeile 100 gesprungen. Vorausgesetzt, Sie haben dort eine solche Routine programmiert.

Die bedingte Verzweigung in Unterprogramme wird ohne ständige IF-Abfragen zur Überprüfung von Fehleingaben möglich: <GKEY Zeilennummer, ASCII-Wert> beachtet nur die Taste, die im Befehlsaufruf angegeben ist, und verzweigt an die angegebene Zeilennummer.

Bildschirmmasken können mit dem Aufruf von <MASKEDIT> erstellt oder editiert werden. Die fertige Bildschirmmaske wird nach Beendigung automatisch in Form von Basic-Zeilen im Speicher abgelegt.

Benötigen Sie für Ihre Bildschirmmaske ein Rechteck als Umrahmung: <RECLOW> erledigt das sehr schnell, die Farbe können Sie ebenfalls als Parameter eingeben. Das

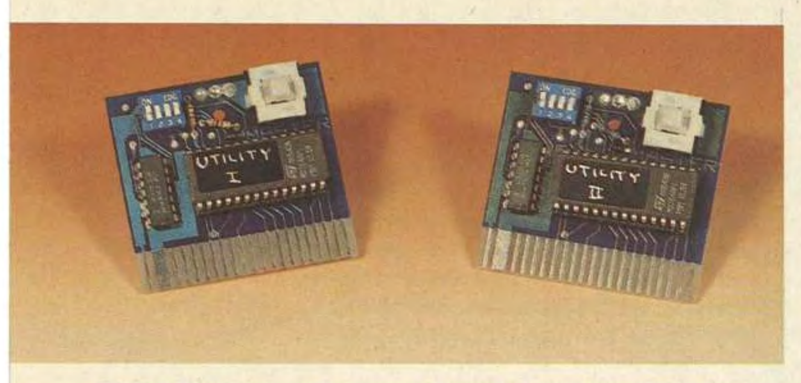


Bild 3. Die beiden Steckmodule von Roßmöller bieten viele Funktionen auf kleinstem Raum.

Scrollen eines rechteckigen Bildschirmausschnitts bewirken Sie mit dem Befehl <SCROLL>. Eine Verschiebung ist in beliebiger Richtung möglich.

Für die Programmierung von Sprites gibt es eine Vielzahl von Befehlen. Mit <BIGSPR> werden beispielsweise die ersten vier Sprites zu einem großen Sprite zusammengefügt. Mit Zusatzparametern legen Sie fest, ob das Sprite im Vordergrund oder Hintergrund erscheinen soll oder ob es sich um ein Multicolor-Sprite handelt. Zusätzlich kann es in X- und Y-Richtung vergrößert werden. Mit dem Befehl <SETBIG> wird das große Sprite auf dem Bildschirm positioniert.

Wollen Sie einen Text an einer bestimmten Zeile und Spalte auf dem Bildschirm ausgeben? Mit <SETCUR> kein Problem: Der Cursor wird entsprechend Ihren Eingaben positioniert.

Was im Standard-Basic nur mit einer längeren Abfrage möglich ist, wird mit Hyper-Basic in einem Befehl erledigt: <INSTR> sucht in einem String einen Teilstring und legt die gefundene Startposition in einer Variablen ab.

Die Soundfähigkeiten des C64 werden von Befehlen wie <WAVE>, <FMOD> und vielen anderen unterstützt.

Ein ganz besonderer Leckerbissen wird mit dem Befehl <DTOOLS> aufgerufen. Mit dem Menü, das nach dem Aufruf auf dem Bildschirm erscheint, können Sie unter verschiedenen Optionen auswählen. Haben Sie unabsichtlich ein Programm auf Diskette gelöscht? Mit den Disk-Tools wird das Retten des Programms kein Problem.

Zusammen mit anderen Tools ist Tool-Basic in den Ausgaben 1/88 und 2/88 des 64'er-Magazins als Listing des Monats veröffentlicht worden.

Ob Sie fantastische Grafiken erstellen, ein professionelles Softscrolling einbauen oder tolle Sprites animieren wollen: Mit den Befehlen dieser speziellen Basic-Erweiterung wird das alles kinderleicht.

Die zahlreichen Befehle lassen sich in drei Kategorien einordnen. Der erste Teil befaßt sich mit der Bildschirmverwaltung, ein zweiter Teil erlaubt eine optimale Spriteverwaltung, der dritte Teil bietet eine Vielzahl von Befehlen für eine sinnvolle Kollisionsverwaltung.

Ob es sich um Kollisionen von Sprites untereinander oder zwischen Sprites und Bildschirminhalten handelt: Tool-Basic läßt kaum Wünsche offen. In Verbindung mit dem Rest des Gesamtpakets lassen sich professionelle Spieleffekte programmieren: »Uridium« läßt grüßen.

Zwei preisgünstige Module, die unter dem Namen »Utility I« und »Utility II« vertrieben werden (Bild 3), bieten – neben weiteren sinnvollen Funktionen – Zusatzbefehle, mit denen Grafikprogrammierung vereinfacht wird.

Die Funktionstasten sind mit nützlichen Befehlen belegt: Anzeige des Directory ohne Datenverlust, OLD-Befehl nach einem versehentlichen NEW oder Reset, Start einer Bildschirmhardcopy. Die Grafikbefehle erlauben ein schnelles Zeichnen von Kreisen, Punkten und Linien. Die Stärken dieser beiden Module liegen sicher nicht in der Basic-Erweiterung, sondern eher in den Zusatzfunktionen wie Sprite-Editor oder Monitor.

Unsere Vorstellung einiger Erweiterungen zum Basic 2.0 des C64 kann natürlich nicht den Anspruch erheben, vollständig zu sein. Es gibt eine Vielzahl anderer Spracherweiterungen, die näher anzuschauen sich lohnt.

Einen Nachteil, den alle Erweiterungen gemeinsam haben, wollen wir nicht verschweigen: Programme, die Sie mit den neuen Befehlen erstellen, funktionieren nur unter der Umgebung der Spracherweiterung. Wenn Sie das Programm weitergeben, müssen Sie Ihre Erweiterung dazulegen. Unter Basic 2.0 meldet sich der C64 sonst nur mit »Syntax error«.

Wer jedoch eigene Anwendungen programmieren will, hat die freie Auswahl. Ein Superpaket und sehr preiswert ist Exbasic Level II. Allen Wünschen gerecht wird Hyper-Basic, das durch seinen Umfang und Zusatztools wie Monitor, Diskettenmonitor und Assembler jeden Basic-Programmierer begeistert. (ef)

Bezugsquellen:
MAGIC-FORMEL V2.0: Grewe Computertechnik GmbH, Richard-Wagner-Str.73, 4350 Recklinghausen, zirka 170 Mark
HYPER-BASIC: Andreas Bude, System Hard- und Software, Bonner Str. 34, 5216 Niederkassel 6, zirka 80 Mark
EXBASIC LEVEL II und UTILITY I und II: Roßmöller Handshake GmbH, Neuer Markt 21, 5309 Meckenheim, jedes einzelne Modul zirka 40 Mark

auto	back	beep
bigspr	bin	blink
callm	case	catalog
centre	check	clearz
clhi	cltext	color
convert	copy	dec
delete	delrem	detekt
diskin	do/loop/exit	dtools
dump	dupear	eob
erchan	erdisk	errinit
erroff	errout	fast
ffoff	fillbs	fillfa
filter	find	fmod
frame	frq	gkey
grepson	grhrdcpy	groff
grpa	grwind	hardcopy
help	help2	hex
hires	huellk	instr
invers	jsaus	jsein
jump	key	killarray
line	lis	lodprg
maskedit	mcvarsprite	mergea
mergee	midch	midstr
mode	modus	move
mulcol	nrm	numeric
off	page	pause
pen	plot	puls
ram	ramvar	reclow
reclr	red	renum
restre	resume	retkey
revtext	room	savprg
screen	scrnld	scrnsv
scroll	search	setbig
setcur	setsid	setspr
settext	sline	slow
small	sort	sound
split	sprein	sprite
sprpar	sprreg	sprtext
string	swapar	swapva
tall	text	trans
varram	varsprite	video
view	vol	wave
winp	wline	woutp
zeilbs		

Tabelle 2. Übersicht der Hyper-Basic-Befehle

DAS KRAFTPAKET:

Über 70 neue Basic-Befehle stellt Ihnen diese mächtige Basic-Erweiterung zur Verfügung. Basic-Listings und abscrollen oder einfaches Belegen der Funktionstasten sind nur zwei kleine Beispiele für die große Leistungsfähigkeit von Exbasic Level II, die das Herz eines jeden Basic-Programmierers höher schlagen lassen.

Basic ist die am weitesten verbreitete Programmiersprache unter den Heimcomputern. Auch der C64 stellt sofort nach dem Einschalten Basic zur Verfügung. Mit Basic 2.0 lassen sich bereits viele Anwendungen programmieren. Wer aber beispielsweise Grafik oder Musik programmieren will, kommt ohne genaue Kenntnisse des Betriebssystems kaum weiter.

Erheblich einfacher wird das Programmieren mit Exbasic Level II (Listing 1). Diese Basic-Erweiterung bietet eine Vielzahl zusätzlicher Befehle und hilfreicher Funktionen, die im normalen Basic 2.0 des C64 oft vermißt werden.

Finden Sie es lästig, die Zeilennummern eines Programmes bei jeder neuen Zeile schreiben zu müssen? Exbasic hilft Ihnen mit einer automatischen Zeilennumerierung. Wollen Sie eine Übersicht aller in Ihrem Programm verwendeten Variablen sehen? Kein Problem für Exbasic.

Strukturiertes Programmieren ist mit Basic 2.0 nicht gerade einfach zu verwirklichen. Für viele Programmieraufgaben bietet sich der Einsatz einer "IF...THEN...ELSE"-Konstruktion an, die im Standard-Basic fehlt. Exbasic hat diese Funktion ebenso integriert wie einige andere Strukturanweisungen.

Die Stärken von Exbasic Level II liegen eindeutig im professionellen Bereich. Mit Exbasic erarbeiten Sie aber beispielsweise leistungsfähige Finanzprogramme, statistische Auswertungen, Balkendiagramme oder andere interessante Anwendungen. (Wer Spiele oder schöne Hires-Grafiken programmieren möchte, wird lieber auf andere Basic-Erweiterungen zurückgreifen.)

Die zusätzlichen Programmierhilfen werden viele Anwender begeistern. Datasettenbesitzer werden von den integrierten schnellen Kassettenlade-Routinen begeistert sein. Besitzer eines Diskettenlaufwerks finden eine Erweiterung integriert, die

dem DOS 5.1 auf der Test/Demo-Diskette verwandt ist. Doch damit genug der Vorrede – kommen wir nun zu einer Darstellung der verschiedenen Befehle von Exbasic. Geben Sie das Listing 1 bitte mit dem MSE (Seite 159) ein und speichern Sie es auf Diskette.

Geladen wird es mit:

```
LOAD "EXBASIC L II",8,1
```

Starten Sie es anschließend mit RUN. Exbasic meldet sich mit einer eigenen Einschaltmeldung. Um die zum Teil sehr komplexen Befehle korrekt beschreiben zu können, werden wir eine einheitliche Syntax vereinbaren. Die nachfolgende Auflistung der Befehle von Exbasic ist zunächst nach Gruppen geordnet. Innerhalb der Gruppen finden Sie die zusätzlichen Befehle alphabetisch sortiert.

Die jeweilige Anweisung ist fettgedruckt. Sind zusätzliche Parameter notwendig oder möglich, werden sie im Anschluß angegeben. Ein kurzes Beispiel soll den Befehl erläutern.

Die verwendeten Zeichen in der Beschreibung bedeuten:
- eckige Klammern: Die eingeschlossenen Parameter sind optional, das heißt sie können gegebenenfalls auch weggelassen werden.
- geschweifte Klammern: Die eingeschlossenen Parameter sind alternativ, das heißt einer der durch Schrägstriche getrennten Parameter wird an dieser Stelle eingesetzt.
- Parameter ohne Klammern: Diese sind bindend und müssen angegeben werden.
- Dollarzeichen: Dieser Ausdruck kennzeichnet einen String.






64er online
64er-online.de
64er-online.net

Kurzinfo: Exbasic Level II

Programmart: Komfortable Basic-Erweiterung
Laden: LOAD "EXBASIC L II",8,1
Starten: Nach dem Laden RUN eingeben.
Besonderheiten: In den C64 eingebaute Floppyspeeder und die ins Programm integrierte Schnellade-Routine arbeiten eventuell nicht zusammen. Verwenden Sie in diesem Fall entweder den LOAD*-Befehl, oder schalten Sie den eingebauten Floppyspeeder aus.
Programmautor: Michael Krause

Editierbefehle

AUTO [zeilennummer[,schrittweite]]
Die automatische Zeilennumerierung, beginnend bei »zeilennummer« in Schritten der »schrittweite«, wird eingeschaltet. Jeweils nach Eingabe einer Programmzeile wird nun die nächste Zeilennummer vorgegeben. Ersatzwert für »zeilennummer« ist 10, beziehungsweise, wenn schon Zeilen eingegeben wurden, die letzte Zeilennummer plus der Schrittweite. Ersatzwert für die Schrittweite ist 10. Der Abbruch dieses Modus erfolgt durch das Drücken von <RETURN> in einer leeren Bildschirmzeile.
Beispiel: AUTO 10,10

DEL bereich
Löscht Programmzeilen. Als »bereich« kann eine Start-, eine End- oder eine Start- und Endzeile angegeben werden.
Beispiele: »DEL 100-«, »DEL -500« oder »DEL 50-300«

DUMP
Gibt die momentanen Werte aller Variablen aus. Starten Sie dazu ein Basic-Programm. Nach Abbruch an beliebiger Stelle listet DUMP alle nichtindizierten Variablen auf.

FIND {text/"text"}[,bereich]
Findet angegebenen Text oder Anweisung(en) im Programm und listet die Zeilen im angegebenen Bereich auf (»bereich« wie bei DEL).
Beispiel: FIND REM findet alle REMs
FIND "Text" findet beliebigen Text

HELP [*]
Gibt mit »*« eine Liste aller Standard-Basic-, sonst aller Exbasic Level II-Befehle auf dem Bildschirm aus.

KEY [{keynummer,ausdruck$}]
Ändert die Belegung der Funktionstasten. In dem String wird die Return-Taste durch »←« repräsentiert.
Standardbelegungen:

F1	LIST	F5	RUN ←
F2	AUTO	F6	GOTO
F3	DUMP ←	F7	FIND
F4	MATRIX ←	F8	MEM ←

Beispiel:KEY 1, "list 10-20←"
Der Pfeil hinter dem Befehl bewirkt, daß der Befehl beim Drücken der Taste sofort ausgeführt wird.

LOAD {*[ausdruck$[,gerätenummer[,sekundäradresse]]]/[ausdruck$]}
»LOAD« lädt Programme mit zirka sechsfacher Geschwindigkeit von Kassette. »LOAD **« wählt das normale Laden.
Beispiel: LOAD*"PROGRAMM"

MATRIX
Gibt die Werte aller Feldvariablen aus (erst sinnvoll nach dem Start und Abbruch oder innerhalb eines Programms).

MEM
Gibt die momentane Speicherbelegung aus.

MERGE {*ausdruck$[,gerätenummer]/[ausdruck$]}
Verbindet zwei Programme miteinander; das erste steht im Speicher, das zweite mit Namen »ausdruck$« wird mit MERGE hinzugeladen. Die Zeilen werden entsprechend ihrer Numerierung in das Programm eingesetzt.
MERGE*"PROGRAMM1",8

RENUM [zeilennummer[,schrittweite]]
Numeriert die Programmzeilen neu durch. Das neue Programm beginnt ab »zeilennummer« mit dem Abstand »schrittweite«. Es werden die Referenzen nach ELSE, GOTO, GOSUB, LIST, RUN, RESTORE, RESUME und THEN korrigiert.
Beispiel: RENUM 10,10

SAVE {*[ausdruck$[,gerätenummer]]/[ausdruck$]}
»SAVE« schreibt Programme mit zirka sechsfacher Geschwindigkeit auf Kassette. »SAVE **« wählt die normale Schreibgeschwindigkeit an.

SPACE [OFF]
Schaltet formatiertes Listen eines Basic-Programms ein: Hinter Befehle werden Leerzeichen zur besseren Lesbarkeit eingefügt.

TRACE {ON/[OFF]}
Aktiviert (»ON«) oder desaktiviert (»OFF«) den TRACE-Modus, in dem die jeweils abgearbeitete Basic-Zeile in den beiden obersten Bildschirmzeilen aufgelistet wird. Nach dem Programmstart mit RUN halten Sie den Programmablauf mit <Control> an. Mit <Commodore> geht es weiter, mit <RUN/STOP> brechen Sie wie gewohnt ab.

VERIFY {*[ausdruck$[,gerätenummer[,sekundäradresse]]]/[ausdruck$]}
»VERIFY« vergleicht das mit »SAVE« gespeicherte Programm mit zirka sechsfacher Geschwindigkeit von Kassette. »VERIFY **« wählt das normale Vergleichen an.
Der Punkt kann an Editierbefehlen angehängt werden und steht für »letzte bearbeitete Zeile«. Beispiel: »LIST.« listet die zuletzt bearbeitete Zeile.

Mathematische Befehle

DEC (ausdruck$)
Liefert den Dezimalwert eines zwei oder vier Zeichen langen Hex-Strings.
A=DEC ("C000"): B=DEC ("9F")

EVAL (ausdruck$)
Berechnet den im String enthaltenen numerischen Ausdruck. Beispiel: A=EVAL ("12*2"): PRINT A liefert 24 als Ergebnis. Nur im Programm möglich.

FRAC (ausdruck)
Berechnet den Nachkommateil eines numerischen Ausdrucks.

HEX$ (adresse)
Erzeugt einen Hex-String des Werts »adresse«.
Beispiel: A$=HEX$(49152)

MIN (ausdruck,ausdruck[,ausdruck...])
Ermittelt den kleinsten Wert der angegebenen Ausdrücke.

MAX (ausdruck,ausdruck[,ausdruck...])
Ermittelt den größten Wert der angegebenen Ausdrücke.
Beispiel: A=MAX(12,-5,78,3,55)

ODD(x)
Prüft den numerischen Parameter »x« auf gerade oder ungerade. ODD ist als Boolesche Funktion anzuwenden, das heißt, es liefert einen Funktionswert, der ausgegeben oder einer Variablen zugeordnet wird. 0 steht hier für gerade, —1 für ungerade Zahl.

RND(x)
Ist eine Erweiterung des Basic 2.0-Befehls. Geben Sie als Parameter »x« einen Wert kleiner als zwei an, so arbeitet RND wie gewohnt. Werte größer als zwei liefern Zufallszahlen zwischen 1 und dem angegebenen Wert.
Beispiele: A=RND(6) Zahl zwischen 1 und 6
A=RND(21)+9 Zahl zwischen 10 und 30

ROUND (ausdruck[,byteausdruck])
Rundet »ausdruck« auf »byteausdruck« Nachkommastellen (Standardwert 0).
Beispiel: A=ROUND (12.5278,2)

**STRING$** (anzahl,{string$/asciicode})
Diese Funktion füllt einen STRING »anzahl« mal mit dem Zeichen, das durch »string$« oder »asciicode« angegeben ist.
Beispiele: A$=STRING$(5,»*«)
A$=STRING$(5,42)
In beiden Fällen enthält A$ die Zeichenkette »*****«, da der ASCII-Code des Malzeichens (»*«) 42 ist.

Ein-/Ausgabe-Befehle

INPUTFORM ["text";]var$[,byteausdruck][,farbe]
Liest eine Tastatureingabe, deren Maximallänge durch »byteausdruck« (Standardwert 79) festgelegt sein kann, in die Variable »var$«; Eingabeaufforderung per »text«.
Beispiel: INPUTFORM "Bitte Namen eingeben:";N$,10,5

INPUTLINE ["text";]var$
Liest eine Eingabezeile ohne Rücksicht auf Kommata und Doppelpunkte von der Tastatur in die Stringvariable »var$«; Eingabeaufforderung per »text«.

PRINT@ bildposition,{ausdruck/ausdruck$}[;]
Gibt den Wert von »ausdruck« beziehungsweise den Inhalt von »ausdruck$«, beginnend an der angegebenen Bildschirmposition, aus. Der Wert für Bildposition kann von 0 bis 999 gehen.
Beispiel: PRINT@ 497,"Hallo"

PRINT USING [#dateinummer,]ausdruck$,ausdruck [,ausdruck..][;]
Gibt die Werte (der Variablen) »ausdruck« in dem »ausdruck$« vorgegebenen Format auf dem Bildschirm oder bei spezifizierter Dateinummer in eine Datei aus. Gültige Formatzeichen in »ausdruck$« sind hierbei <#> (Ziffer), <*> (Ziffer als <*> wenn gleich 0), <+> (Vorzeichen immer ausgeben), <-> (Vorzeichen ausgeben, wenn Zahl negativ), <,> oder <.> (Dezimalkomma oder -punkt). Andere Zeichen werden identisch übernommen.
Beispiel: »PRINT USING "DM ***,##+",-8.272« führt zur Ausgabe »DM **8,27-«.

Diskettenbefehle

\> oder @
Zeigt den Fehlerstatus der Diskettenstation an.

\>$ [text]
Gibt das gesamte beziehungsweise den durch »text« spezifizierten Teil des Inhaltsverzeichnisses der Diskette auf dem Bildschirm aus, ohne das im Speicher befindliche Programm zu löschen.

/ name
Lädt ein Programm von Diskette.

↑ name
Lädt ein Programm von Diskette und startet es.

← name
Speichert ein Programm auf Diskette.

DOS-Kommando:	Funktion:
c:nameneu=namealt	Dateien kopieren.
i	Diskette initialisieren.
n:diskname,id	Diskette formatieren.
r:nameneu=namealt	Dateien umbenennen.
s:name	Datei löschen.
v	Diskette reorganisieren.

\>dos-kommando
Sendet ein DOS-Kommando an die Diskettenstation.

Strukturanweisungen

IF ausdruck **THEN** block **[ELSE** block]
Ist die zwischen »IF« und »THEN« stehende Bedingung erfüllt, so wird der nach »THEN« folgende Teil der Programmzeile bearbeitet, andernfalls werden die hinter »ELSE« folgenden Anweisungen abgearbeitet.

ON ausdruck **RESTORE** zeilennummer[,zeilennummer...]
Der angegebene »byteausdruck« bestimmt, auf die wievielte der Zeilennummern sich der »GOTO/GOSUB/RESTORE«-Befehl beziehen soll. Sind nicht genügend Zeilennummern vorhanden, so wird der Befehl ignoriert.

ON ERROR GOTO {zeilennummer/0}
Initialisiert eine Fehlerbehandlungsroutine, die bei Auftreten eines Fehlers während der Ausführung eines Programms ab jetzt angesprungen wird, durch Angabe der ersten Zeilennummer der Routine. Zeilennummer 0 schaltet dies aus. Die Variable »EL« enthält die Nummer der fehlerhaften Programmzeile und »EC« die Nummer des Fehlers gemäß folgender Tabelle:

Code	Bedeutung	Code	Bedeutung
00	Modul	16	Out of memory
01	Too many files	17	Undef'd statement
02	File open	18	Bad subscript
03	File not open	19	Redim'd array
04	File not found	20	Division by zero
05	Device not present	21	Illegal direct
06	Not output file	22	Type mismatch
07	Not input file	23	String too long
08	Missing filename	24	File data
09	Illegal device number	25	Formula too complex
10	Next without for	26	Can't continue
11	Syntax	27	Undef'd function
12	Return without gosub	28	Verify
13	Out of data	29	Load
14	Illegal quantity	30	Resume without error
15	Overflow	31	Format

Tabelle. Die Fehler-Codes von Exbasic Level II

RESUME [{zeilennummer/**NEXT**}]
»RESUME« beendet die Fehlerbehandlungsroutine. Ohne Parameter wird zu der zum Fehler führenden, mit NEXT zu der darauf folgenden Anweisung und bei Angabe einer Zeilennummer zu dieser Programmzeile verzweigt.

Variablenbearbeitung

EXEC var$
Führt den Inhalt des Strings »var$« als Basic-Kommando aus.

INSTR (ausdruck1$,ausdruck2$[,byteausdruck])
Sucht »ausdruck2$« in »ausdruck1$« ab Position »byteausdruck« (Standardwert 1). Ergebnis ist die Position der Übereinstimmung; 0 entspricht nicht gefunden.

SWAP {var,var/var$,var$}
Tauscht die Werte zweier Variablen gleichen Typs aus.

VARPTR (variablenname)
Diese Funktion gibt die Adresse an, an der eine bestimmte Variable im Arbeitsspeicher vom Betriebssystem abgelegt worden ist. Beispiel: A=13:PRINT VARPTR (A) liefert die Speicheradresse für A.

Soundbefehle

ADSR stimme,welle,a,d,s,r[,pulsweite]
Wählt die ADSR Parameter (a=attack, d=decay, s=sustain, r=release) für den Synthesizer. Zulässige Werte: stimme: 1 bis 3, attack: 0 bis 15, decay: 0 bis 15, sustain: 0 bis 15, release: 0 bis 15, Pulsweite: 0 bis 4095. Als Wellenform kann Dreieck (17), Sägezahn (33), Rechteck (65) mit wählbarer Pulsweite zwischen 0 und 4095 sowie Rauschen (129) ausgewählt werden.

PAUSE byteausdruck
Wartet »byteausdruck« geteilt durch 60 Sekunden.

PLAY stimme,tonhöhe[,stimme,tonhöhe...]
Spielt eine oder mehrere Noten.

VOLUME byteausdruck
Bestimmt die Lautstärke der durch PLAY gespielten Noten (Bereich 0=leise bis 15=laut).

Grafikbefehle

BORDER farbnummer
Legt die Bildschirmrandfarbe fest.

CEEK (bildposition,{c/s})
Fragt den Inhalt (Zeichencode beziehungsweise Farbnummer) einer Bildschirmposition ab, wobei »s« das Zeichen selbst oder »c« seine Farbe auswählt.
Beispiel: X=CEEK(3,S)

COKE bildposition,{s/c},ausdruck
Ändert das Zeichen (»s«) beziehungsweise die Farbe der Bildschirmposition gemäß »ausdruck« (Zeichencode beziehungsweise Farbcode).
Beispiel: COKE 888,65

CURSOR farbnummer
Legt die Cursorfarbe fest (0 bis 15).

GROUND farbnummer
Legt die Hintergrundfarbe des Bildschirms fest.

HARDCOPY
Druckt momentanen Bildschirminhalt auf einem mit der Geräteadresse 4 angeschlossenen Drucker (MPS-Serie) aus.

LETTER [OFF]
Schaltet zwischen Groß-/Kleinbuchstaben (LETTER) und Groß/Grafik-Modus um.

LOCK {[ON]/OFF}
Verriegelt die Groß/Kleinbuchstaben-Umschaltung.

POINT (x,y)
Liefert -1, wenn Viertelpunkt gesetzt, sonst 0.

RESET x,y
Löscht einen Punkt der Viertelpunkt (Lowres)-Grafik.

SET (x,y)[,farbnummer]
Setzt einen Punkt der Viertelpunktgrafik mit der Farbe farbnummer.
Beispiel: SET (5,7)

SPACE bildbereich[,byteausdruck[,farbnummer]]
Füllt den angegebenen Bildbereich mit dem durch »byteausdruck« angegebenen Zeichen oder löscht ihn, wenn »byteausdruck« weggelassen wird.
Beispiel: SPACE 0,0,39,24,65,1
Die Werte setzen sich so zusammen: Spalte links oben, Zeile links oben, Spalte rechts unten, Zeile rechts unten, Bildschirmcode (das Zeichen A im Beispiel), Farbwert (schwarz im Beispiel).


```
Name : exbasic 1 ii       0801 2564
-----------------------------------
0801 : 0c 08 c3 07 9e 32 30 36  8c
0809 : 32 ff 00 00 00 78 a0 c5  0d
0811 : b9 46 08 99 fe 00 88 d0  d6
0819 : f7 84 01 84 ac 84 ad a2  0e
0821 : 04 b5 aa d0 02 d6 ab d6  f8
0829 : aa ca ca d0 f4 b1 ae 91  c0
0831 : ac a9 0c c5 ae a9 09 e5  96
0839 : af 90 e4 a9 00 85 ae a9  d9
0841 : 80 85 af 4c ff 00 a2 de  41
0849 : b1 ac 20 b4 01 9d 32 01  b7
0851 : e8 d0 f5 a9 01 85 60 a9  65
0859 : f9 85 5f a2 04 20 12 02  cf
0861 : f0 29 c9 0f d0 15 20 10  90
0869 : 02 d0 0b a2 04 20 12 02  78
0871 : 69 0f 85 5d 90 05 a2 08  3b
0879 : 20 12 02 20 b2 01 f0 71  01
0881 : 20 bb 01 c6 5d d0 f4 c6  56
0889 : 5e 10 f0 20 10 02 d0 27  d2
0891 : a9 02 85 61 a2 05 20 12  c0
0899 : 02 38 a5 ae e5 5d 85 5d  11
08a1 : a5 af e5 5e 85 5e b1 5d  30
08a9 : e6 5d d0 02 e6 5e 20 bb  0c
08b1 : 01 c6 61 d0 f1 f0 a4 20  01
08b9 : 10 02 d0 1a a9 03 85 61  cd
08c1 : 20 10 02 d0 cf a2 0d 20  0a
08c9 : 12 02 69 20 85 5d a5 5e  d1
08d1 : 69 00 85 5e 90 c3 e8 20  72
08d9 : 12 02 4a d0 04 69 04 d0  d6
08e1 : dd b0 07 20 12 02 69 06  bf
08e9 : d0 d4 a2 08 20 12 02 90  89
08f1 : cd a9 37 85 01 58 4c e2  db
08f9 : fc b3 ac e6 ac d0 02 e6  fe
0901 : ad 60 91 ae e6 ae d0 02  44
0909 : e6 af 60 a2 01 86 5c 84  f2
0911 : 5d 84 5e c6 60 d0 09 a9  25
0919 : 08 85 60 20 b2 01 85 5f  08
0921 : 06 5f 26 5d 26 5e c6 5c  35
0929 : d0 e9 a7 5d 60 f9 33 18  56
0931 : 81 5e fe c3 c2 cd 38 30  f5
0939 : 93 2b 2b 20 45 58 42 41  d3
0941 : 53 49 43 20 4c 45 56 45  e1
0949 : 4c 20 49 49 20 2f 20 56  ca
0951 : 36 34 2e 33 20 86 0d 0d  18
0959 : 20 0f 89 50 52 4f 47 2e  86
0961 : 20 42 59 20 4d 2e 4b 52  15
0969 : 41 55 53 45 61 00 f0 d8  5e
0971 : 7c 98 07 0e 4b 45 d9 0d  1d
0979 : 4d 45 4d 4f 52 59 ba 42  05
0981 : 59 54 45 53 8d 00 a5 a4  79
0989 : 52 41 4d ba 56 41 52 49  72
0991 : 41 42 4c 24 ba 82 02 59  05
0999 : 39 22 53 54 44 4e 47 33  7d
09a1 : 46 52 45 45 ba 52 1f 32  29
09a9 : 55 4d 45 20 57 49 54 48  9c
09b1 : 4f 55 d4 46 4f 52 4d 41  e8
09b9 : d4 00 0a 14 1e 28 32 3c  f7
09c1 : 46 50 04 05 06 03 87 80  69
09c9 : 95 80 00 00 f9 83 73 81  2b
09d1 : 7c a5 1a a7 06 83 d8 83  82
09d9 : 08 af 41 01 4c 48 b2 00  fb
09e1 : f4 d0 23 2a 2c 2e 2b 2d  87
09e9 : 20 7e 7c e2 7b 61 ff ec  60
09f1 : 6c 7f e1 fb 62 fc fe a0  60
09f9 : 82 00 f8 f7 20 64 6f 79  8e
0a01 : 62 f8 f7 e3 20 65 74 75  44
0a09 : 61 f6 ea e7 a0 91 6f 92  16
0a11 : 06 2a 24 25 8c 8a a7 89  a9
0a19 : 8d e0 ef 9b a4 a7 a9 af  13
0a21 : b0 ef a8 30 39 30 34 38  4f
0a29 : 33 4c c0 83 00 4c ca 83  b7
0a31 : 20 50 fd 20 15 21 a3 fc  e0
0a39 : d0 5b ff 58 20 e0 e3 a9  ae
0a41 : 00 8d 83 02 85 37 a0 80  bf
0a49 : 84 38 8c 84 02 a9 09 20  6f
0a51 : 2d e4 20 3e 81 4c 9d e3  79
0a59 : 20 23 9d a2 13 bd ae 80  a1
0a61 : 9d 00 03 ca 10 f7 a2 06  70
0a69 : bd 0e 81 95 7c af 8f 91  e7
0a71 : f8 a9 0a a2 00 8d fe 02  81
0a79 : 8e ff 02 85 fd 86 fe 85  53
0a81 : be 86 bf 85 19 86 1a 60  12
0a89 : a5 7f 29 f7 85 7f 20 f2  f2
0a91 : 88 86 7a 84 7b 20 73 00  12
0a99 : aa f0 ed a2 ff 86 3a 90  c9
0aa1 : 09 20 20 8e 20 00 82 4c  39
0aa9 : 06 83 20 6b a9 aa 20 f8  49
0ab1 : 8b 29 08 f0 51 e0 2a d0  57
0ab9 : 05 00 4e f0 ca a4 14 c4  1c
```

VPLOT ausdruck[,farbnummer]
Zeichnet vertikale Balkengrafik mit der Balkenlänge »ausdruck« (Bereich 0 bis 200).
Beispiel:
10 PRINT TAB(4);
20 VPLOT (SIN(X)+1)*120:X=X+.1:GOTO.

HPLOT ausdruck[,farbnummer]
Zeichnet horizontale Balkengrafik mit der Balkenlänge »ausdruck« (Bereich 0 bis 200). Beispiel: Ersetzen Sie in Zeile 20 »vplot« durch »hplot«.

Systembefehle

BASIC
Verläßt EXBASIC LEVEL II und kehrt zum Standardbasic zurück, ohne das aktuelle Programm zu löschen (Reaktivierung mit »?USR(0)«).

DEF USR (adresse)
Richtet den Einsprung-Vektor der Basic 2.0-Funktion USR(x) auf den 16-Bit-Wert »adresse«.
Beispiel: DEF USR (828) richtet den USR-Vektor auf ein Maschinensprache-Programm im Kassettenpuffer.

DEEK (adresse)
Doppelbyte-PEEK. Ergebnis ist der aus »adresse« (LSB) und Adresse+1 (MSB) errechnete Wert: (Inhalt von »adresse«)+256*(Inhalt von adresse+1).

DISPOSE {**CLR/RETURN/NEXT**[var]}
Schließt offene »FOR« und »GOSUB«- Schleifen. »DISPOSE CLR« schließt alle offenen, »RETURN« das innerste »GOSUB« und »NEXT« die entsprechende »FOR - NEXT«-Schleife.

DOKE adresse,ausdruck
Zerlegt den Doppel-Bytewert »ausdruck« (im Bereich von 0-65535) in LSB und MSB und speichert ihn an adresse (LSB) und an adresse+1 (MSB).
Beispiel: DOKE 1,826 entspricht:
POKE 1,58 und
POKE 2,3

HIMEM adresse
Stellt die obere RAM-Grenze ein und führt »CLR« aus.

LETTER [{**ON/OFF**}]
LETTER ON oder LETTER ohne Parameter schaltet auf den Groß-/Kleinschrift-Zeichensatz um, LETTER OFF auf den Groß-/Grafik-Zeichensatz.

LOCK [{**ON/OFF**}]
Verriegelt oder erlaubt, je nach Parameter, das Umschalten der Zeichensätze mittels der <SHIFT>- und der <Commodore>-Taste.

RESTORE [zeilennummer]
Ist eine Zeilennummer angegeben, so wird der DATA-Lesezeiger auf diese Zeile gestellt.

SEC byteausdruck
Wartet »byteausdruck« Sekunden.

Um ein Basic-Listing auf- und abzuscrollen, benutzen Sie die entsprechenden Cursor-Tasten.

Wir wünschen Ihnen viel Spaß bei der Anwendung der neuen Befehle, die ein komfortables Programmieren erlauben.

(Michael Krause/ef)

Exbasic Level II als Modul

Auf der Programmservice-Diskette finden Sie unter dem Namen »EXBASIC MODUL« eine Version von Exbasic, die in ein EPROM gebrannt werden kann. Wer die dafür erforderlichen Hardware-Zusätze nicht besitzt, kann das fertige Modul für 39,95 Mark bei folgender Adresse bestellen:

Roßmöller Handshake GmbH
Neuer Markt 21
5309 Meckenheim
02225/2061


```
0ac1 : be a6 15 8a e5 bf 90 0c  20
0ac9 : 84 80 2a f0 86 20 12 92  0d
0ad1 : 90 03 4c 3e 92 20 39 93  f4
0ad9 : a5 15 48 a5 14 0f 7a eb  4b
0ae1 : bf 85 14 86 15 20 13 a6  25
0ae9 : a0 00 c8 b9 ff 00 99 76  46
0af1 : 02 d0 f7 b0 03 a9 20 2c  c6
0af9 : a9 2a 0d 84 c6 68 f0 0f  1d
0b01 : 30 15 a4 15 84 3c c8 f0  b7
0b09 : 81 a6 14 86 3b 04 df 93  2e
0b11 : 40 a2 a4 a9 99 d0 64 a6  00
0b19 : 7a a0 04 84 0f bd 00 02  58
0b21 : 10 07 c9 ff f0 55 e8 d0  26
0b29 : f4 85 08 c9 22 f0 76 24  e7
0b31 : 0f 70 48 c9 20 d0 0a 49  07
0b39 : a1 40 f0 e9 a9 5e d8 3a  d9
0b41 : c9 3f f0 ce c9 30 90 04  28
0b49 : c9 3c 90 2e 20 c5 82 a0  96
0b51 : 00 84 0b ca e8 e6 22 d0  9f
0b59 : 02 e6 23 01 ff 91 06 38  cc
0b61 : f1 22 f0 f1 c9 80 d0 44  4a
0b69 : a4 23 c0 a0 b0 09 a5 0b  e3
0b71 : c9 16 90 64 69 c5 2c 05  75
0b79 : 0b a4 71 e8 c8 99 fb 01  9b
0b81 : b9 e2 f0 50 38 e9 27 f0  43
0b89 : 12 c9 13 f0 04 c9 49 d0  b8
0b91 : 02 85 0f 6b 55 d0 83 85  7c
0b99 : 08 00 40 d8 f0 da c5 08  d9
0ba1 : f0 d6 00 4d 70 e8 d0 f0  1a
0ba9 : a6 7a e6 0b b1 22 08 01  f6
0bb1 : ea 13 28 10 f4 b1 c8 9c  6a
0bb9 : a5 23 c9 a0 b0 0a a9 a0  b9
0bc1 : 85 23 a9 9d 85 0e 80 01  c3
0bc9 : 27 97 a4 4c 09 a6 69 01  dc
0bd1 : 90 9d 84 71 a9 9e 02 e7  e6
0bd9 : c7 c8 4d 86 7a 60 20 2c  ac
0be1 : a8 a4 7b c0 02 f0 06 a6  df
0be9 : 7a 86 3d 84 3e a0 00 b1  d3
0bf1 : 7a f0 07 c9 ef d0 4e 20  dd
0bf9 : 3b a9 a0 02 41 18 f0 41  8c
0c01 : c8 93 85 39 85 3b 27 6c  c3
0c09 : 3a 85 3c 20 fb a8 05 21  74
0c11 : a0 c7 a6 3a e8 f0 1d 34  79
0c19 : 1a 01 f0 03 20 ea 94 a7  4b
0c21 : e9 22 02 f0 0e a6 7a a4  03
0c29 : 7b 8e 04 03 8c 05 03 ba  bf
0c31 : 8e fd 02 20 79 00 20 40  db
0c39 : 83 4c d2 82 4c 4b a8 c9  3d
0c41 : 3a f0 c9 4c 08 af c9 cb  ac
0c49 : b0 26 c9 17 b0 04 69 22  77
0c51 : d0 11 e9 80 b0 09 c9 a7  fe
0c59 : f0 03 4c a5 a9 a9 0f c9  4a
0c61 : 23 b0 e0 0a a8 b9 35 9e  c0
0c69 : 48 b9 34 0c 4c 02 0f b7  9d
0c71 : 08 e9 a3 c9 4d b0 cb 90  3a
0c79 : e9 4c 12 a8 c9 b7 d0 0a  d4
0c81 : 20 96 83 8c 11 03 8d 12  e2
0c89 : 03 60 c9 08 8b 84 0c e1  01
0c91 : 0d 38 e0 4c b3 b3 1c 32  2a
0c99 : 20 92 9a 20 8a ad 4c f7  e4
0ca1 : b7 ad 8d 02 01 a8 87 14  6e
0ca9 : 2d 28 d0 fb 0b c0 cb cf  23
0cb1 : 04 f0 f6 6f 13 4c c2 02  7b
0cb9 : e7 83 02 a9 3a c9 3a b0  54
0cc1 : 0a c9 20 f0 07 38 e9 30  10
0cc9 : 28 d0 60 04 e5 a9 00 85  a9
0cd1 : 0d 20 1e e0 c9 bb d0 03  56
0cd9 : 4c c3 85 c9 ef 90 09 6f  28
0ce1 : f9 05 e9 a4 4c 5d 83 20  6a
0ce9 : c6 83 4c 8d ae 8a 10 97  e4
0cf1 : 74 a4 a4 3a c8 f0 4c d0  0f
0cf9 : 32 de 3c e0 10 f0 46 86  74
0d01 : 49 b3 e6 10 fd 09 04 85  23
0d09 : 7f ae fd 02 9a ad 0e 03  f4
0d11 : ae 0f 03 38 86 60 20 c7  8a
0d19 : a8 a4 49 20 a2 b3 20 65  7d
0d21 : 84 a5 3a a6 39 85 62 86  32
0d29 : 63 a2 90 38 20 49 bc a2  8d
0d31 : 4c 20 67 84 1e 1f 32 a9  ef
0d39 : 74 a0 84 8d 14 03 8c 15  86
0d41 : 03 e0 1f b0 03 4c 3a a4  57
0d49 : 8a 0a aa bd 6a 80 85 22  40
0d51 : bd 6b 80 4c 45 a4 a2 43  f8
0d59 : a9 45 85 45 86 46 20 e7  9a
```

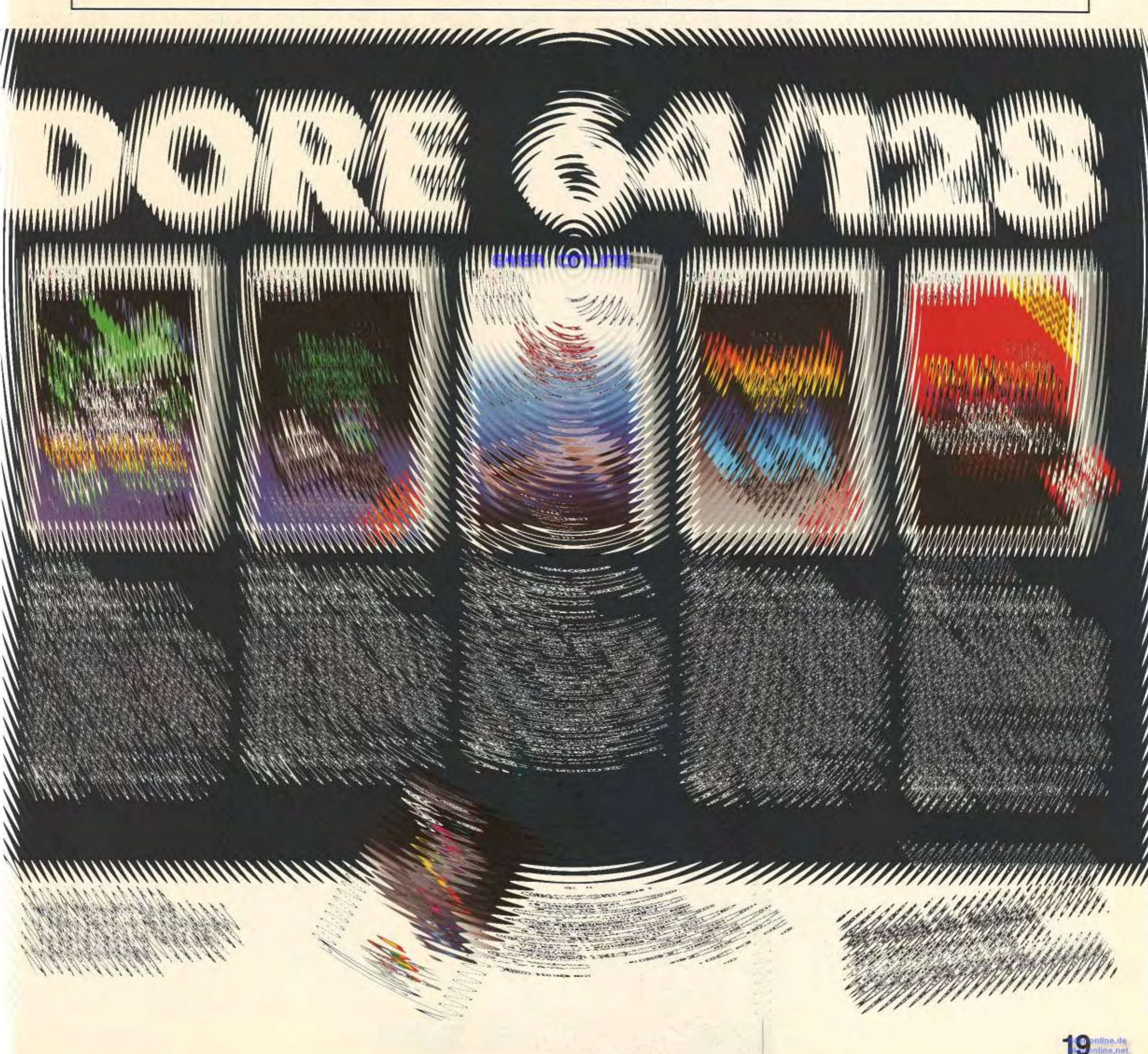
Listing 1. »EXBASIC L II« erweitert das Basic des C64. Bitte mit dem MSE (Seite 159) eingeben.

```
0d61 : b0 aa 4c d4 bb a5 3e 05  00
0d69 : cc 81 06 6e 31 ea a9 31  b9
0d71 : a0 ea 00 17 f9 50 a9 93  59
0d79 : 9d 05 01 a9 84 9d 06 01  5e
0d81 : d0 e7 a9 91 20 16 e7 20  74
0d89 : 07 96 20 d7 aa a6 84 20  11
0d91 : 8c e8 a5 83 85 d3 46 cf  1b
0d99 : 4c cd e5 a2 02 2c a2 00  a6
0da1 : 7e aa 01 86 83 20 98 9a  56
0da9 : 20 9e b7 e0 50 90 02 a2  f9
0db1 : 4f 86 84 20 f1 b7 20 95  f1
0db9 : 9a e0 32 ef ab a7 31 86  18
0dc1 : 64 8a 4a aa bd f0 ec 85  74
0dc9 : 61 85 ae b5 d9 29 03 0d  5c
0dd1 : 88 02 85 62 20 e9 e9 a5  4c
0dd9 : 84 4a a8 b1 61 a2 0f dd  06
0de1 : ca 80 f0 03 ca d0 f8 86  ac
0de9 : 63 a9 01 46 64 90 02 0a  11
0df1 : 0a 46 84 90 01 0a 85 84  d1
0df9 : a6 83 f0 09 e0 02 f0 13  c6
0e01 : 49 0f 25 63 2c 05 63 aa  55
0e09 : bd ca 80 91 61 ad 86 02  1f
0e11 : 91 ae 60 a5 84 d0 f0 02  5d
0e19 : a9 ff 20 3c bc 4c 79 00  66
0e21 : c9 82 d0 21 27 49 d0 04  8c
0e29 : a0 00 04 13 30 8b b0 85  5a
0e31 : 49 84 4a 20 8a a3 16 31  d3
0e39 : 30 ad 8a 18 69 12 aa 9a  ec
0e41 : 01 e3 10 c9 9c f0 1a c9  be
0e49 : 8e d0 c4 77 e3 49 7a 70  b3
0e51 : 9a c9 8d 00 07 e0 a8 ba  c3
0e59 : 00 02 d8 07 aa 2c a2 fa  fe
0e61 : 9a 1e 61 c0 9e ad 20 17  81
0e69 : 8c 89 13 f1 c0 a7 d0 03  4f
0e71 : 19 a5 a5 61 f0 17 90 40  7d
0e79 : 90 a8 a0 a8 c9 2e d0 e9  c0
0e81 : 8b 8d 68 68 62 ba 4c 09  39
0e89 : 83 a2 01 15 42 30 0f c8  b4
0e91 : c9 8b 10 70 a1 03 ae 90  40
0e99 : f0 ca d0 ed 53 65 d0 cd  1f
0ea1 : 81 06 bc f1 ae 20 2b bc  a5
0ea9 : f0 02 0d a5 19 9a e0 20  bd
0eb1 : cc bc a5 61 c9 82 b0 45  6f
0eb9 : be e0 a2 f8 a0 02 20 d4  f3
0ec1 : bb 20 0e a9 55 30 28 ba  32
0ec9 : a9 bc a0 b9 20 67 b8 00  50
0ed1 : 8d 4c 8d ad 20 fa ae 01  34
0ed9 : 43 78 98 a4 47 20 91 b3  36
0ee1 : 20 f7 ae 4c 79 01 49 c1  7a
0ee9 : a5 66 48 0e b0 66 20 0c  6c
0ef1 : bc 20 cc 66 53 b8 68 60  1b
0ef9 : 5c 91 d0 36 7f 23 e4 a9  11
0f01 : 48 a2 eb 8d 8f 02 8e 90  ab
0f09 : 02 a9 3e a2 81 8d 11 03  93
0f11 : 8e 12 03 68 68 4c ae a7  69
0f19 : a0 0b 84 d3 20 cd bd a0  83
0f21 : 11 1c 0b ec b9 50 80 08  d1
0f29 : 29 7f 20 47 ab c8 28 10  c5
0f31 : f3 60 01 ec a0 8d ad 0a  73
0f39 : c2 00 c6 b4 a9 da a0 80  38
0f41 : 20 a2 bb 20 14 bb 03 e0  92
0f49 : 42 30 01 85 12 20 1b b0  84
0f51 : 4c 79 5a 90 80 6d 0c 03  ac
0f59 : ca b6 8f ad a5 64 48 a5  02
0f61 : 65 48 20 fd ff 01 a3 b6  b6
0f69 : f0 e3 85 9c 86 c1 84 c2  4e
```



```
0f71 : 68 a8 68 20 aa d3 d5 85  f7
0f79 : 9b 86 b5 84 b6 a2 00 80  d7
0f81 : 8e 64 2c d0 07 20 9b b7  b6
0f89 : 8a f0 c0 ca 86 a5 05 0f  dc
0f91 : ae a5 9b 38 e5 9c 90 2c  de
0f99 : e5 a5 90 28 69 00 85 a7  76
0fa1 : a5 a5 18 65 b5 85 b5 90  4b
0fa9 : 02 e6 b6 04 7b cb b5 d1  dd
0fb1 : c1 d0 0b c8 c4 9c 90 f5  16
0fb9 : a4 a5 c8 4c a2 b3 e6 a5  9a
0fc1 : c6 a7 a0 d2 f8 60 f3 e6  0d
0fc9 : b5 d0 df e6 b6 d0 db 60  dd
0fd1 : d0 fd a0 03 a9 0c 0e db  13
0fd9 : 42 87 a0 11 a9 02 8a 57  ad
0fe1 : 19 a9 04 a2 02 29 5c 23  45
0fe9 : a9 06 a2 04 a4 60 2a 01  b6
0ff1 : 07 08 29 f8 fc 32 a9 08  7e
0ff9 : a2 06 48 20 53 86 68 a8  11
1001 : b9 2b 00 38 f5 2b 85 87  35
1009 : b9 2c 00 f5 2c a6 87 4c  46
1011 : 46 86 91 49 04 11 40 f0  d4
1019 : 38 54 04 1d 46 c0 20 f7  fb
1021 : b7 a5 15 c9 10 b0 04 a0  01
1029 : 03 d0 1d 66 8d ea 08 71  dc
1031 : ba 76 84 49 85 4a 00 24  64
1039 : 0a 78 a0 01 91 49 88 a5  98
1041 : 14 3c 58 02 32 19 a8 28  a9
1049 : 82 b1 14 aa 88 14 a8 8a  df
1051 : 0c 38 40 e3 b4 10 07 a9  41
1059 : f0 a0 80 0d 28 40 8c 04  1a
1061 : 08 a8 41 74 8e 8a 82 1d  1e
1069 : e8 f4 ae a5 0d 30 0c 20  ee
1071 : bf b1 a5 64 d0 26 a5 65  9e
1079 : 4c e6 87 20 82 b7 f0 1c  00
1081 : 80 db f3 f0 22 85 83 68  37
1089 : 20 7d b4 a8 f0 09 a5 83  9f
1091 : 88 91 62 c0 00 d0 f9 20  41
1099 : ca b4 0a bf 8f c9 22 d0  a9
10a1 : 0b 20 bd ae a9 3b 20 ff  f6
10a9 : ae 20 21 ab 20 a6 b3 90  4c
10b1 : ad 1e 4c 8f ad 60 20 01  d3
10b9 : 88 03 97 c3 cf ff c9 0d  5f
10c1 : f0 06 9d 00 02 e8 d0 f3  ae
10c9 : a9 00 a0 02 54 d0 aa 20  91
10d1 : 89 b4 4c da 1b ed e8 a0  29
10d9 : 9e 28 f0 07 04 c7 e4 a2  00
10e1 : 9e a0 a0 86 83 84 84 a0  78
10e9 : 00 20 a2 83 a2 00 b1 83  0a
10f1 : f0 c1 c0 79 8e d2 ff c8  32
10f9 : 16 97 c1 84 e8 28 10 ed  c8
1101 : 20 3f ab e8 e0 0a 90 f8  5b
1109 : b0 de c8 1b 5b 37 85 38  b4
1111 : 4c 60 a6 a9 00 2c a9 80  75
1119 : 85 bd 01 54 16 e5 a2 f3  4b
1121 : a1 48 06 03 78 7c 29 f0  ba
1129 : 13 01 f0 39 a9 a4 65 5b  2c
1131 : bc 45 bd 30 e1 10 e6 a1  83
1139 : 04 46 50 4c f7 1b 01 03  60
1141 : b2 f0 1d a2 10 85 83 86  4f
1149 : 84 9f 1e 6c a2 2d a0 00  c8
1151 : 88 d0 fd ca d0 fa c6 84  23
1159 : d0 ef c6 83 a5 83 d0 e3  c4
1161 : 60 02 ac 20 1d 02 00 a0  15
```

Listing 1. »EXBASIC L II« Fortsetzung



```
1169 : 2a b6 a0 04 b1 83 aa b1  dc
1171 : 47 91 83 8a 91 47 88 09  3a
1179 : e8 33 68 b4 9b 48 20 85  33
1181 : 87 68 aa 71 e0 a9 00 91  94
1189 : 49 bd 16 03 65 82 0b 34  96
1191 : b4 e4 60 03 e5 f9 a9 01  07
1199 : d0 a0 c0 01 02 11 09 1f  15
11a1 : 68 a8 8a 99 7c 20 3e 89  08
11a9 : a0 05 91 49 9e 28 06 4c  35
11b1 : 5a 87 20 9e 9b 0a 47 85  dd
11b9 : 29 8d 05 87 60 b1 83 20  bd
11c1 : 62 89 c4 b6 85 c8 c1 05  9f
11c9 : 23 20 60 c9 3a 08 29 0f  f4
11d1 : 28 90 02 69 08 60 53 c2  45
11d9 : 06 d8 20 ad 03 01 09 08  76
11e1 : 72 1b 86 aa f0 ec 4a b0  d9
11e9 : e9 c9 03 b0 e5 1d a9 05  86
11f1 : 20 4d 89 85 86 5b a8 a5  fc
11f9 : 86 4c ad 87 b0 77 de a9  97
1201 : 02 a6 15 f0 01 0a 06 81  35
1209 : 21 a0 00 a5 15 14 d2 bd  e8
1211 : 89 a5 14 95 68 4c f8 87  00
1219 : 48 4a 67 20 d2 89 91 62  e9
1221 : c8 68 29 0f 8a 52 60 18  37
1229 : 69 f6 01 37 d7 06 69 3a  fc
1231 : 60 a4 44 99 3c 03 e6 44  28
1239 : 60 68 68 a9 00 20 dc 89  a4
1241 : a9 3c a0 03 20 1e ab 01  34
1249 : 80 10 87 70 d7 aa a9 3b  b1
1251 : a0 00 d1 7a 58 43 06 f8  5f
1259 : e1 4c cc 8a a4 44 c4 43  eb
1261 : f0 d6 b1 49 a2 06 dd c3  ab
1269 : 80 f0 0a 40 83 40 1c 9c  d0
1271 : d0 e7 84 bd a9 2c 20 fc  84
1279 : 89 06 4f d9 dd bd a2 00  6b
1281 : 86 61 86 63 ae 00 01 a4  fe
1289 : 44 b1 49 c9 2b d0 0d e0  61
1291 : 20 d0 02 a2 2b 8a e6 00  91
1299 : 94 10 d4 f8 2d f0 f5 60  7c
12a1 : aa a5 63 c9 2a f0 07 a9  c9
12a9 : ca 40 03 aa a9 30 2b 84  7b
12b1 : 0b 90 c2 60 18 d5 83 a6  4d
12b9 : 61 bd 01 01 c9 2f 90 09  c4
12c1 : a4 e0 e6 61 c6 83 d0 ee  65
12c9 : 60 a2 00 0a e5 b0 12 00  e8
12d1 : 58 b0 2a d0 04 85 63 fe  1e
12d9 : e2 23 44 74 e8 c8 d0 ea  da
12e1 : 86 66 a0 00 00 8a da 08  92
12e9 : c9 45 f0 08 a4 f1 98 85  d9
12f1 : 65 05 61 40 a4 bd 45 0f  a5
12f9 : 44 40 45 0c 05 04 51 40  66
1301 : f0 36 2f 56 c8 d0 e8 a2  9f
1309 : 20 4c 37 a4 20 a6 0e 3c  9a
1311 : 10 05 03 11 07 f4 b8 86  8f
1319 : 13 20 18 e1 0e 42 f2 ec  17
1321 : 8a 4c b5 ab 21 82 41 d8  91
1329 : 85 43 86 60 04 0b 07 80  b3
1331 : 44 20 08 8a 20 7d 8a d0  92
1339 : 13 a6 66 f0 1e 20 52 c1  07
1341 : 66 80 e8 f1 dc a9 30 99  6f
1349 : 3b 03 d0 11 a5 66 38 e5  96
1351 : 65 f0 05 90 90 20 51 07  df
1359 : 00 80 0e d0 80 e6 90 f0  9a
1361 : 02 e6 61 12 47 2c f0 04  13
1369 : 0b 0a 90 2f 20 de 89 04  aa
1371 : 61 00 42 b0 25 c5 66 b0  f4
1379 : 0d 01 41 a0 0c fc 7f 20  52
1381 : 5c 8a f0 14 8a 20 68 8a  41
1389 : c9 39 f0 0c bd 02 01 c9  30
1391 : 35 90 05 a6 44 fe 3b 03  54
1399 : 20 34 8a 4c fe 8a 02 3e  c8
13a1 : 10 4a 7c 70 85 83 8a 29  f4
13a9 : 07 85 84 a5 d1 18 65 d3  64
13b1 : 85 85 a5 d2 69 00 3c 29  96
13b9 : d7 9a a0 00 0c db 6d 1a  8f
13c1 : a9 a0 20 fa 9a a5 85 38  7f
13c9 : e9 28 f9 33 b0 09 c6 86  27
13d1 : a5 86 cd 88 02 90 0b ca  a4
13d9 : d0 e6 a6 84 bd df 80 d3  db
13e1 : d3 c5 d5 86 37 58 42 ab  74
13e9 : 60 f0 18 a1 90 89 e0 13  fb
13f1 : b0 bd 10 e3 a8 20 f8 a8  c1
13f9 : a5 5f a4 60 38 e9 01 4c  f2
1401 : 24 a8 4c 1d a8 c9 f0 d0  6e
1409 : 31 20 fe 43 89 07 84 10  75
1411 : 00 ed a5 14 05 15 d0 07  3e
1419 : 0f 90 90 f9 85 7f 60 02  ad
1421 : 98 90 11 a5 5f a6 60 8d  c2
1429 : 0e 03 8e 0f 03 b1 5b 09  7c
1431 : 02 04 01 28 40 41 0c 48  4a
1439 : 34 a9 c0 08 c9 89 63 dc  a3
1441 : 8c d0 12 c6 65 d0 04 68  50
1449 : 4c 4c 83 01 48 40 12 04  93
1451 : 06 e9 60 ee 68 55 e1 08  0a
1459 : 02 10 ec 40 b0 c3 79 76  a2
1461 : 06 8a d4 f0 eb c5 83 b0  5c
1469 : 02 85 83 14 cb a5 83 a8  da
1471 : 18 65 01 04 dc 90 0a a9  ca
1479 : 11 20 16 0f 6c 1c 81 95  db
1481 : 70 a9 00 65 d2 85 86 98  17
1489 : f0 08 a9 20 14 ef 88 d0  70
1491 : f8 84 84 a9 64 55 e0 20  d6
1499 : e4 ff f0 fb a4 84 c9 14  f6
14a1 : d0 0c c0 00 f0 f1 e3 ea  ab
14a9 : 10 e2 c9 5f f0 d3 c9 0d  77
14b1 : d0 0f 40 30 00 4f 14 d3  91
14b9 : aa 98 f0 84 aa 4c 30 88  5b
14c1 : c4 83 b0 ce aa c9 de d0  63
14c9 : 3b d6 c2 99 00 02 8a 30  ee
14d1 : 08 c9 20 90 bd 29 [illegible] b0  5b
14d9 : 2b 85 51 c0 b3 09 40 07  c6
14e1 : 34 c8 d0 a4 20 b5 b1 12  dd
14e9 : 41 c0 64 a5 65 95 54 b0  0e
14f1 : 03 88 30 08 e8 e0 19 d0  e1
14f9 : f3 04 e6 60 86 d6 69 28  c9
1501 : 85 d3 20 6c e5 20 8f 9a  d8
1509 : 8a 23 ed 02 59 b0 9d aa  c7
1511 : 4c d1 8a 20 9c 83 c9 02  fe
1519 : b0 dd aa fa 8b 20 85 83  99
1521 : 98 48 01 5a c0 20 24 ea  dc
1529 : 68 aa 4a b6 a4 d3 05 83  54
1531 : f0 15 01 4b e2 84 a9 a0  90
1539 : 91 d1 20 08 9b c8 c0 28  0f
1541 : f0 10 c4 84 d0 f0 81 ab  ed
1549 : 4d aa bd e7 80 d6 20 a9  ea
1551 : 11 4c 47 ab 20 dd 91 4c  9f
1559 : 16 08 85 7f 68 68 2e 31  a9
1561 : 61 a9 ff 4c bf 81 90 82  6f
1569 : 2e f8 21 35 4b d0 06 28  a6
1571 : 5c 59 a6 20 71 8d 28 d4  f5
1579 : a2 8b 8d 05 1a 20 0b 8e  d1
1581 : 4c a3 a8 01 e4 20 2c bc  63
1589 : 26 91 aa a5 7f e0 91 d0  be
1591 : 07 09 80 85 7f 4c f8 a8  7d
1599 : 29 9c 01 f0 f5 0b a0 44  31
15a1 : 17 08 f1 a3 4a 4b 59 e0  d3
15a9 : 28 b0 d4 e4 49 90 d0 86  64
15b1 : ac 6d c3 19 b0 c7 e4 4a  99
15b9 : 90 c3 86 ad a2 20 7a 8e  b4
15c1 : 03 00 0a 18 0b 23 c0 a6  64
15c9 : 4a ca e8 20 f0 e9 4f 7c  4b
15d1 : 8e a4 49 88 c8 a5 83 91  00
15d9 : d1 a5 87 91 f3 c4 ac d0  4b
15e1 : f3 e4 ad d0 e5 4c 6c e5  0a
15e9 : 18 a2 9d 69 e4 90 01 e8  8f
15f1 : 86 4a 60 60 0a b9 70 0b  07
15f9 : a5 3b a6 3c c2 c0 83 f5  99
1601 : 0c 21 64 b0 6b a9 85 7f  e6
1609 : 5e c9 2f f0 16 c9 3e f0  c0
1611 : 3c c9 40 f0 38 c9 5e f0  8d
1619 : 10 c9 5f d0 77 20 97 8e  f4
1621 : 4c 59 e1 4c 39 6f 37 c8  99
1629 : 62 a9 00 a6 2b a4 2c 20  fe
1631 : 56 8e 86 2d 98 4c 57 a6  ac
1639 : 20 d5 ff b0 0a 20 b7 ff  da
1641 : 29 bf f0 a8 4c 9c e1 4c  65
1649 : f9 e0 d3 06 48 00 23 c3  01
1651 : 68 d0 17 a9 08 20 b4 ff  71
1659 : a9 6f 20 96 ff 20 a5 ff  2c
1661 : 03 e8 10 37 51 f6 4c ab  19
1669 : ff c9 24 f0 23 76 37 b1  9a
1671 : 67 4c ea f3 17 18 70 88  3c
1679 : 84 b7 89 f0 0a 86 90 86  7d
1681 : 93 e8 86 bb 18 bd bc e4  ce
1689 : 85 ba 60 a5 9a 85 a4 a5  ec
1691 : b8 85 a5 20 2b 8f 20 d5  d5
1699 : f3 a2 05 e0 02 e1 60 c0  6d
16a1 : 0b 8a 22 8f 11 2c 47 aa  51
16a9 : 06 60 3d 93 a2 ff e8 bd  ea
16b1 : 00 01 f9 a9 20 e8 9d ff  a5
16b9 : 00 a8 f0 05 b7 93 f0 f4  b0
16c1 : 20 ab ff a6 a4 e0 03 a5  34
16c9 : a6 a5 20 c9 ff a2 00 e8  6a
16d1 : bd c5 20 ca f1 d0 f7 8a  6d
16d9 : 4b 08 69 68 20 2f f3 20  1b
16e1 : ed f6 d0 07 80 3c b8 38  9c
16e9 : 4c 34 a8 20 ab 8e a2 03  3d
16f1 : d0 9e 08 64 a4 90 f0 86  3f
16f9 : 85 25 6e 60 85 b9 4c 42  94
1701 : f6 06 89 0e 06 82 4c 33  2b
1709 : 20 33 ae 05 30 21 98 ea  53
1711 : a2 00 68 95 61 e8 e0 05  6b
1719 : d0 f8 0b aa 34 e2 ba c6  50
1721 : 83 10 f9 20 49 94 84 68  4b
1729 : 8f 20 fe 40 50 84 08 92  fe
1731 : a0 86 98 0e 36 0c a4 a1  96
1739 : 1f 99 b6 7c 10 93 a4 2d  ed
1741 : a5 2e 84 3f 4c dd 96 20  55
1749 : ac 96 d0 eb a1 55 0a 20  1f
1751 : d4 e1 a5 0a a6 2d a4 a5  83
1759 : 02 60 70 20 33 a5 fc 08  10
1761 : 2d f0 4e c8 29 64 85 14  a7
1769 : a1 a6 e4 09 03 21 05 a8  56
1771 : 65 88 10 f8 0e c8 d6 ef  a0
1779 : d0 f8 18 c8 84 0b a5 da  d1
1781 : 5a 65 0b 85 58 a4 2e 84  6e
1789 : 5b 90 01 3e 58 59 20 b8  77
1791 : a3 a5 31 a4 32 85 2d 84  f5
1799 : 2e a4 0b 88 b9 00 02 91  b4
17a1 : 5f 01 55 30 a9 20 60 a6  47
17a9 : 8b de b8 d0 69 a9 04 85  ea
17b1 : b8 85 ba a9 00 ac b0 7e  35
17b9 : 43 84 44 85 83 85 b7 85  4e
17c1 : b9 20 c1 e1 a6 b8 c0 40  eb
17c9 : 46 19 03 bc a0 f0 3d 78  6c
17d1 : d2 43 29 80 c5 2c 00 1f  9b
17d9 : 48 49 80 09 12 7b 40 47  94
17e1 : 44 85 4a 29 3f 06 4a 24  35
17e9 : 4a 10 02 09 80 70 1b 40  56
17f1 : 50 82 ea 57 d0 d5 98 18  76
17f9 : 65 06 43 25 4c 60 44 4d  4a
1801 : 3d 30 ca d0 be 20 cc ff  43
1809 : a9 04 4c 91 f2 60 a0 49  41
1811 : a5 93 d0 02 a0 59 4c 2f  59
1819 : f1 a5 17 6c 85 8b 84 8c  10
1821 : c4 30 8b 40 c5 2f b0 59  33
1829 : 69 02 46 79 89 85 22 84  aa
1831 : 23 20 b7 90 20 e1 90 8a  cd
1839 : 10 06 20 e5 90 4c a8 90  40
1841 : 98 30 1a f5 90 54 a0 fb  5d
1849 : 90 0b 92 a5 8b a4 8c 2c  20
1851 : 7d 60 90 c4 c8 b0 c1 63  9b
```

```
1859 : 82 01 dc b8 8b aa 21 08  4d
1861 : 09 12 a8 4a f0 03 ae 29  1b
1869 : 8a 10 04 a9 25 d0 46 98  55
1871 : d2 24 d0 3f 60 c9 30 a9  da
1879 : 3d d0 37 8e 32 aa c8 10  7a
1881 : 22 48 a3 95 b3 4c f8 9f  24
1889 : 20 a6 bb 26 48 1a 91 a0  8d
1891 : 02 8f c2 25 88 a5 1c 24  1f
1899 : 95 70 26 f0 0a b1 24 03  d3
18a1 : 17 c4 26 d0 f6 a9 22 4c  9c
18a9 : 2a a0 a6 30 a5 2f 85 8b  d4
18b1 : 86 8c e4 32 22 de 31 b0  3c
18b9 : ae a0 04 69 05 0c 3e b2  f5
18c1 : 85 8d 86 8e b1 8b 0a a8  71
18c9 : 65 8d da c1 86 c2 88 84  8d
18d1 : 8f a9 00 99 05 81 72 53  35
18d9 : fa 30 32 a4 8f e5 fd 98  5e
18e1 : aa fe 06 02 d0 03 fe 41  70
18e9 : b9 0d 76 d1 8d d0 06 c8  0a
18f1 : 87 90 14 a9 00 a4 fd 99  4b
18f9 : 2c a0 99 06 03 62 d7 a5  8a
1901 : c1 a6 c2 4c 23 91 03 65  e5
1909 : 20 a4 8f a9 28 16 d1 b9  82
1911 : 04 02 be d4 84 fd 20 cd  b5
1919 : 87 58 bc a4 fd 88 88 10  f6
1921 : ea a9 29 20 de 90 00 24  e9
1929 : 23 af c7 23 00 d6 28 10  f2
1931 : 07 10 05 a9 02 d0 11 c8  33
1939 : 30 4c 30 13 f5 06 05 40  22
1941 : 2c 47 fb 68 03 18 65 c1  27
1949 : 85 c1 15 d1 c2 81 0f 3c  1b
1951 : 4c 54 91 15 dc e1 2f b0  c9
1959 : 13 64 7c 30 08 a6 14 a5  15
1961 : 15 86 19 85 1a 86 be 85  8c
1969 : bf 28 f0 1a de 85 b0 fb  90
1971 : 9c 27 a5 14 a6 15 08 45  4a
1979 : 62 05 15 f0 33 28 d0 e6  47
1981 : 60 18 a5 be 6d 2c 85 be  fa
1989 : a5 bf 6d ff 3c e0 bf b0  94
1991 : 02 c9 fa 60 20 56 81 95  28
1999 : cc 2b a6 2c 85 5f 86 72  7c
19a1 : 62 a0 92 f0 36 0c e3 f6  57
19a9 : 70 10 92 01 09 4a 10 45  94
19b1 : a9 a5 19 a6 1a 88 65 07  d2
19b9 : 40 03 24 a0 03 a5 bf 12  18
19c1 : 71 0a a5 be 10 01 81 00  88
19c9 : 28 f8 d0 ed f0 d5 20 f8  8f
19d1 : 93 00 f0 e2 d3 0a a9 b1  94
19d9 : 5f f0 f3 14 bd e9 08 a5  c6
19e1 : 0f 49 ff 85 0f d0 12 24  4d
19e9 : 0f 30 0e c9 8f f0 df a2  12
19f1 : 08 dd f8 80 f0 15 87 b9  7f
19f9 : 38 c8 d0 db 0d 03 5f aa  01
1a01 : c8 11 06 86 5f 85 60 42  cc
1a09 : 60 18 98 65 57 7a 85 5a  5c
1a11 : a6 60 04 95 38 7b 86 5b  cb
1a19 : 67 0c 90 0a c9 ab f0 39  1c
1a21 : c9 a4 d0 d0 f0 33 c1 b9  ae
1a29 : e1 1a 93 a5 5a a6 5b 85  04
1a31 : 7a f4 a2 00 a0 00 83 e7  b6
1a39 : 3c f0 0f 48 00 1e f4 03  85
1a41 : 20 63 93 68 91 7a e8 d0  37
1a49 : ec 20 4b e0 89 52 c8 ce  20
1a51 : 93 f0 f6 aa 38 a5 7a e5  d6
1a59 : 5f a8 8a 6b 41 a1 80 29  92
1a61 : 9f 9d c9 a4 f0 99 aa 4c  f5
1a69 : 7d 92 0d 02 c0 e0 24 86  64
1a71 : 25 d5 1a 24 c5 15 d0 18  04
1a79 : 88 9c 62 14 00 0f bf a6  2f
1a81 : be c2 8e e9 38 4c df bd  62
1a89 : 0c 52 a0 01 71 e2 d0 05  82
1a91 : a9 ff aa d0 e5 e4 42 00  8d
1a99 : 51 f0 85 24 4c 2a 93 86  ba
1aa1 : 83 3a cd 86 58 84 59 e8  66
1aa9 : d0 01 c8 e4 37 98 e5 38  09
1ab1 : 91 a1 35 35 a4 84 3c 35  d1
1ab9 : 2d a0 01 a2 00 a1 58 91  5d
1ac1 : 58 a5 58 d0 02 c6 59 c6  65
1ac9 : 11 c5 7a a5 59 e5 7b b0  24
1ad1 : ea 83 ba a4 85 5a 86 5b  bc
1ad9 : 01 a4 94 5a d0 04 a6 83  6b
1ae1 : 88 60 aa 88 e8 a8 b0 06  f8
1ae9 : c8 d0 07 e8 e0 d0 01 ca  26
1af1 : 88 98 a0 00 91 5a 48 8a  10
1af9 : c8 b5 85 5b 68 85 5a 4c  1d
1b01 : a3 93 a5 2d 00 4f b1 2e  1a
1b09 : c6 2d a5 7a a6 7b 85 58  2b
1b11 : 86 59 a0 01 6d 4b 58 81  22
1b19 : 58 e6 58 16 ca 30 59 07  5f
1b21 : a3 2d 00 7a ed 2e 90 ec  17
1b29 : b0 a1 a9 2b 06 03 0c 38  93
1b31 : 60 32 48 48 69 2e 09 11  14
1b39 : 2c 29 ee 8c 98 35 c9 c7  31
1b41 : a9 80 2c 25 a8 5d 91 02  9a
1b49 : 5e 27 0e 8c 8e 13 e9 c8  0b
1b51 : d0 e0 60 69 86 9c d4 30  d8
1b59 : f7 21 e3 f3 0b 00 30 43  50
1b61 : 34 47 0c c6 43 a9 56 8c  09
1b69 : e3 90 ca ed df a5 7a 85  25
1b71 : 0f c3 07 0c d2 38 f6 0a  84
1b79 : 55 60 8c 95 c9 00 d0 c7  43
1b81 : a5 58 a6 59 0a 25 b0 05  be
1b89 : 80 f0 ec b7 a0 02 a5 14  8c
1b91 : d1 5f a5 15 c8 f1 5f 90  d9
1b99 : 5a f2 c0 98 45 43 85 86  41
1ba1 : f8 8e e3 83 e2 67 86 49  60
1ba9 : ff 56 d0 35 24 86 30 31  48
1bb1 : a6 0f 84 44 23 fe 30 16  a0
1bb9 : 07 8b 12 d1 5f f0 0a 00  ea
1bc1 : 6c f1 1a a5 43 f0 [illegible] d0  d7
1bc9 : 14 3e 63 e5 81 f4 36 02  2e
1bd1 : 84 85 20 d8 9d 05 a4 8a  e5
1bd9 : 73 94 a4 44 8c b8 12 28  6f
1be1 : 60 49 f3 fb 18 8a a6 60  93
1be9 : 69 02 85 2d 81 05 b8 2e  da
1bf1 : 60 a5 d3 48 a5 d5 62 d6  62
1bf9 : 19 62 d4 48 24 ab 06 f0  1b
1c01 : 10 29 02 80 6f 0c 00 06  9a
1c09 : f3 44 10 2c 29 ef 85 7f  cf
1c11 : 7a 9c 10 d0 10 6d e1 65  b6
1c19 : 1e 89 63 0d 00 0c de 16  7e
1c21 : f0 f0 20 66 e5 16 83 73  62
1c29 : c3 a4 83 84 d3 b1 d1 49  54
1c31 : e4 2d 68 85 d4 68 aa 90  d3
1c39 : 10 21 d5 94 d3 60 06 41  bd
1c41 : 40 f0 03 7f 86 12 66 30  9d
1c49 : 0c 20 83 5a 05 2e 10 2f  f2
1c51 : 25 86 5a 96 0f 91 20 b1  84
1c59 : 95 e6 5a 32 11 5b a5 2d  1b
1c61 : c5 50 e8 a5 2e c5 5b d0  5d
1c69 : e2 a5 58 85 2e ab 85 2e  97
1c71 : 4c 33 a5 81 25 06 03 2c  d7
1c79 : 7d 20 d4 94 03 bc 12 04  34
1c81 : 03 15 05 16 91 90 0f c9  80
1c89 : 2d 2d 07 ab d0 46 81 54  72
1c91 : 9c 25 b0 f5 16 40 c0 48  a2
1c99 : 24 86 00 0e 18 00 91 e0  4c
1ca1 : 68 f0 19 01 41 c0 00 a1  45
1ca9 : a8 3f 94 b0 1c 00 19 40  d3
1cb1 : d0 17 02 07 40 15 f8 03  05
1cb9 : a5 5a c5 58 a5 5b e5 59  87
1cc1 : 90 03 a9 00 60 a9 01 a8  e6
1cc9 : 20 a4 d3 91 d1 c4 d5 b0  5e
1cd1 : f5 c8 6e 16 c9 bd a5 39  1c
1cd9 : a6 3a c1 f6 c3 00 13 3c  ec
1ce1 : d3 a6 d6 34 dd a0 a0 03  af
1ce9 : 84 0b 84 49 02 47 49 4f  5b
1cf1 : b0 08 17 c3 06 a5 0b 81  a0
1cf9 : 7c 48 0b c8 a6 60 5c 91  77
1d01 : 5f 81 39 8e c5 7a d0 0c  cc
1d09 : e4 7b d0 08 80 13 08 1a  d5
1d11 : 47 a6 30 0f 24 0b 30 cf  94
1d19 : 92 f6 89 cb 69 c5 20 9f  87
1d21 : 96 30 c0 25 b1 c0 b3 74  7d
1d29 : bc c9 db b0 ef 20 76 7c  aa
1d31 : bc af a2 a0 86 23 a2 9e  cb
1d39 : 86 22 84 49 aa a0 00 0a  de
1d41 : f0 10 ca 10 0c 27 9a e3  1a
1d49 : b1 22 10 f6 30 f1 c8 94  cd
1d51 : 30 1b 02 8b c0 d0 f6 38  e0
1d59 : e9 46 a2 9e 06 13 e8 86  8b
1d61 : e7 d2 c9 ac aa 79 04 04  48
1d69 : f9 c5 a0 01 2c cf 8c 93  28
1d71 : 38 90 4a 41 0c 93 0a ba  a7
1d79 : c3 68 e1 4c 65 e1 98 a5  85
1d81 : a5 3f c5 2b 99 c6 40 e5  39
1d89 : 2c 18 78 a3 3f 85 40 20  b5
1d91 : 17 f8 b0 d1 20 af f5 20  22
1d99 : 4f 99 20 c5 97 11 86 f9  86
1da1 : 00 63 cf b6 20 fd 97 b0  cf
1da9 : f0 a4 b6 91 b2 c8 c0 1e  7c
1db1 : d0 f0 a0 63 20 2f f1 a2  16
1db9 : 00 03 0e 31 b2 0d f3 d1  ec
1dc1 : bb f0 05 c4 b7 b0 01 e8  a5
1dc9 : 9e 3c ed 8a d0 c8 20 69  f9
1dd1 : 90 00 d8 44 3f e6 06 20  a3
1dd9 : 6f 99 4c 04 f7 20 f2 97  24
1de1 : 85 c1 4a 34 c2 ab 65 3f  fe
1de9 : 85 ae 08 51 e1 28 65 40  67
1df1 : 85 af a6 3f a4 40 f0 13  16
1df9 : 38 a5 ae e5 c1 59 95 86  b6
1e01 : c1 a5 af e5 c2 54 84 c2  a4
1e09 : b0 08 a5 08 3f a5 c1 40  d0
1e11 : 80 5c 38 b1 84 b6 00 fd  fe
1e19 : a6 93 d0 0c a6 b6 d0 0a  b6
1e21 : d1 3f f0 06 e6 21 60 02  8c
1e29 : 91 3f e6 3f 37 c6 40 02  aa
1e31 : 7e 60 ae d0 df a5 40 c5  5d
1e39 : af d0 d9 01 c2 01 45 a0  71
1e41 : a5 b1 05 b6 d0 16 31 06  66
1e49 : 0c 46 f0 b0 a3 8f 04 36  fe
1e51 : 4c a1 e1 20 8d e1 85 52  8d
1e59 : 08 73 13 3a a2 1c 8c d4  0e
1e61 : f5 59 a9 ff 8d 05 dc 85  ed
1e69 : b0 20 21 98 26 b0 ad 01  25
1e71 : dc c9 7f f0 13 a5 b0 c9  e4
1e79 : 16 d0 ee 06 01 98 26 df  a1
1e81 : 1c f0 f7 60 01 70 61 16  65
1e89 : e1 62 55 90 29 81 12 f0  cc
1e91 : 50 a2 07 0e 43 d7 a0 0b  41
1e99 : cd a5 d7 45 b0 85 d7 ca  03
1ea1 : 10 f2 75 0c 2a 45 d7 4a  ca
1ea9 : a5 b0 45 b1 85 b1 b0 60  97
1eb1 : 78 ad 0d dc 5e d9 0b ad  1b
1eb9 : 05 dc 10 f4 c9 fe b0 f0  08
1ec1 : 90 03 56 83 0a a9 01 8d  e6
1ec9 : 2c a9 11 8d 0e dc 00 00  88
1ed1 : 68 0c 03 60 02 dc 11 c5  e3
1ed9 : c3 56 80 b3 9c 20 38 f8  fb
1ee1 : 0e 51 40 8f f6 ab dd a9  31
1ee9 : 16 20 cf 98 c6 b6 d0 f7  6b
1ef1 : a9 2a 48 40 28 34 a5 c4  0e
1ef9 : b7 b0 04 a9 70 c1 58 b1  08
1f01 : bb 96 a4 a5 02 90 3a ea  48
1f09 : 00 33 54 e6 b6 10 f7 a9  b4
1f11 : 24 96 a5 3f 4a 39 40 28  91
1f19 : e4 41 a3 42 4a 50 86 b1  74
1f21 : 30 21 9c 07 60 85 1c 41  0f
```

Listing 1. »EXBASIC L II« Fortsetzung

```
1f29 : d0 ed 08 52 db 42 90 e7  1e
1f31 : a5 b1 85 1a e8 4c 6f 99  35
1f39 : 78 85 13 01 12 4c 86 d7  a6
1f41 : a2 08 a5 09 d7 1e 7a 06  d6
1f49 : b0 a0 e2 b0 02 a0 be 8c  51
1f51 : 06 dc a0 19 a5 01 09 08  a7
1f59 : 20 05 99 29 f7 5f 28 ca  38
1f61 : 30 16 d0 dc 06 d7 b8 50  0f
1f69 : df 48 a9 01 2c 0f dc d0  47
1f71 : fb 68 8c d1 85 01 60 e8  b1
1f79 : 60 20 0e e2 c2 ab b3 a2  67
1f81 : 04 b5 2a 95 3e ca d0 f9  0f
1f89 : 14 cf 6f b7 86 b9 e8 86  3e
1f91 : ba 20 d7 f7 20 06 16 57  89
1f99 : e2 4c dc 16 99 85 40 84  6b
1fa1 : 3f d4 b0 42 84 41 60 78  83
1fa9 : a2 ef 07 dc a9 09 8d 0f  d7
1fb1 : dc 13 79 e5 b6 a9 a7 8d  a4
1fb9 : 04 dc a5 01 29 df 85 01  5e
1fc1 : a9 ef 8d 11 d0 60 a9 33  04
1fc9 : c0 45 ce fe 85 c0 01 a4  6b
1fd1 : 6d 20 46 1b d4 a1 58 60  c0
1fd9 : 4a 02 0d 64 22 f8 02 0d  00
1fe1 : 70 30 79 41 0c 83 a5 18  93
1fe9 : ea 2c a0 b2 2d d0 ef 36  ed
1ff1 : 42 74 06 a9 ff 85 14 85  ac
1ff9 : 15 a0 01 71 71 45 84 d7  d0
2001 : 1e c8 06 f8 17 c5 26 04  64
2009 : e4 14 f0 02 b0 31 84 49  ad
2011 : 20 50 30 38 c0 e4 38 a0  c2
2019 : e0 73 7a c8 f0 18 b1 5f  c0
2021 : d0 15 20 fa 95 84 a1 0f  05
2029 : 0c 1c b5 61 d0 b3 60 30  69
2031 : 10 d4 c6 d1 0e 21 30 cd  dd
2039 : 69 c5 48 38 91 19 83 89  a1
2041 : 1b c1 46 bd 48 a6 7f 10  5e
2049 : 05 4c e0 c9 9a 68 e4 c6  f4
2051 : e5 0e 4c 34 3e 01 68 b6  d2
2059 : 9c 03 20 ba 9a a4 49 4c  63
2061 : e0 99 46 5c a0 01 48 3a  d3
2069 : 0f a4 b7 d0 04 d7 31 c0  17
2071 : 50 90 ca d0 b4 3d 7c b8  6f
2079 : b1 bb 29 ed 4a a6 b7 c8  5a
2081 : 98 9d f1 70 68 aa a5 7a  da
2089 : 48 a5 7b 48 a9 02 84 7a  3d
2091 : 85 7b 8a f0 0f c6 69 6a  36
2099 : 20 73 e5 85 88 9a 96 1b  8b
20a1 : a6 ad 68 60 1b 86 7a 60  d5
20a9 : a9 2c 2c a9 b2 21 29 8e  9f
20b1 : 28 a0 fc 7a ee 4c 73 96  04
20b9 : 21 45 ac 9a 88 ac 6c 0c  b3
20c1 : 03 d0 dd 68 c0 61 0a c9  84
20c9 : 44 a6 c9 a3 1f f4 30 cb  39
20d1 : fb b3 f0 b0 10 4c 3f ab  b0
20d9 : a2 05 2c 1f e4 93 01 81  df
20e1 : f0 f3 2c 09 60 ae 86 02  91
20e9 : c1 ce 95 1b 9b 86 87 60  a7
20f1 : 20 eb b7 a5 14 1e 82 71  c8
20f9 : 15 18 6d 59 03 86 26 1e  da
2101 : 88 df 9a 68 91 02 73 f3  0b
2109 : a5 86 29 03 09 d8 85 f4  f3
2111 : 19 f0 12 3e 4b 1e 9b 8d  1e
2119 : 21 d0 b3 94 20 d0 90 00  ed
2121 : 62 48 95 d0 f7 ca 30 05  c7
2129 : 8a c9 10 90 ef 11 94 19  ba
2131 : d1 34 86 02 3c fa a7 20  79
2139 : 10 00 79 aa ad 88 02 a0  65
2141 : ff e0 43 d0 06 a0 0f 80  3e
2149 : 15 ac 18 65 15 85 15 16  65
2151 : 41 aa 31 14 a8 10 fa 05  b7
2159 : 20 a2 88 38 04 04 f0 43  9e
2161 : 19 1c 04 d0 08 04 f9 41  af
2169 : a2 1f c9 85 70 e1 fb e9  98
2171 : 81 82 de f4 16 08 ad 04  ea
2179 : 03 ac 05 03 20 76 ab 28  29
2181 : f0 06 7c f8 63 08 4b 2d  b0
2189 : 83 4c 82 8d 00 80 18 b6  56
2191 : 45 e1 10 b0 09 8a 60 20  88
2199 : a1 9b 8e 18 d4 ea a9 a9  4d
21a1 : d4 85 4a 6f f2 38 ca 30  35
21a9 : f3 e0 03 b0 ef 8a bc 4d  c4
21b1 : 80 84 49 60 46 cf 90 db  af
21b9 : a4 ce ae 87 02 4c 18 ea  1a
21c1 : a6 d6 d0 59 20 c9 9b a2  36
21c9 : ff ec bb f0 4f b5 d9 10  75
21d1 : f7 20 7c 9c b0 f2 85 f2  2a
21d9 : 0e 45 f0 28 c5 2b 21 69  d8
21e1 : 2c f0 34 85 bb ca 86 bc  e9
21e9 : a0 8b 21 4a bb aa d0 fa  2b
21f1 : 64 c5 5f d0 f6 c9 d2 60  f3
21f9 : d0 ef 88 17 05 64 bb 85  33
2201 : 5f a5 bc 1d 42 34 fc 70  a0
2209 : 9c 6e 61 c8 c9 bb 4e ca  97
2211 : 02 c6 cc 4c 48 eb a6 d6  5f
2219 : e0 18 d0 f2 00 47 48 19  25
2221 : ca 30 ea 04 51 f9 c0 11  48
2229 : 46 f4 01 14 a0 e6 14 e0  00
2231 : 87 15 10 6f 58 02 f0 d1  32
2239 : a9 8d 81 90 80 1a 48 a4  5e
2241 : d3 f0 07 f0 bf 17 03 88  3e
2249 : d0 f9 a9 18 85 d6 84 d3  4c
2251 : d0 b6 bc f0 ec 84 7a 0b  bc
2259 : c1 8e 7b 10 6a 13 14 c7  61
2261 : 05 e9 2f 8c 50 9c ee 92  83
2269 : 02 01 b1 47 f0 0f 4e 7a  f7
2271 : 30 bd ef ec 85 ac b5 d8  5f
2279 : 20 c8 e9 30 ee 20 ff e9  41
2281 : a2 17 b5 da 29 7f b4 d9  8d
2289 : c5 87 95 da ca 10 f1 a5  13
2291 : d9 8f 04 48 d9 60 a0 02  63
2299 : 7e 40 83 d7 8c d4 85 d8  4a
22a1 : 85 c7 60 d0 44 5c 44 25  be
22a9 : 5e 5b bc 9a 80 8a 09 30  18
22b1 : 0b e7 1c 2c 01 9e e5 17  07
22b9 : 9d 86 49 a2 0a 86 d8 b9  ec
22c1 : e4 9d 25 5a 10 9d c8 ca  af
22c9 : d0 f4 d4 30 7b c9 49 e8  4b
22d1 : e0 09 d0 cb 60 c9 0d 89  7f
22d9 : a1 2d 5f 2c 2f 75 20 c9  21
22e1 : 91 d0 22 67 07 ea c3 21  69
22e9 : 4e a9 7f a2 9d 16 e3 4c  f3
22f1 : a6 03 a4 06 23 bc 30 18  0c
22f9 : 13 09 b0 f6 81 6e 9f bd  21
2301 : 9b 80 20 ff 8d ea 85 49  bd
2309 : 0a 5a 71 f0 df c9 0b b0  95
2311 : db 85 43 c6 12 c9 5f 8f  65
2319 : b8 a9 0d 91 49 c8 c0 0a  0d
2321 : f0 c7 a9 00 2d 8a f3 90  77
2329 : e9 ad ca 02 d0 2b a6 cc  76
2331 : 12 51 a5 cb c9 07 d0 12  0b
2339 : e6 cc ee 0c 21 f4 a6 01  19
2341 : d0 49 bc 10 9c 4c d5 9b  a2
```

ROCKUS

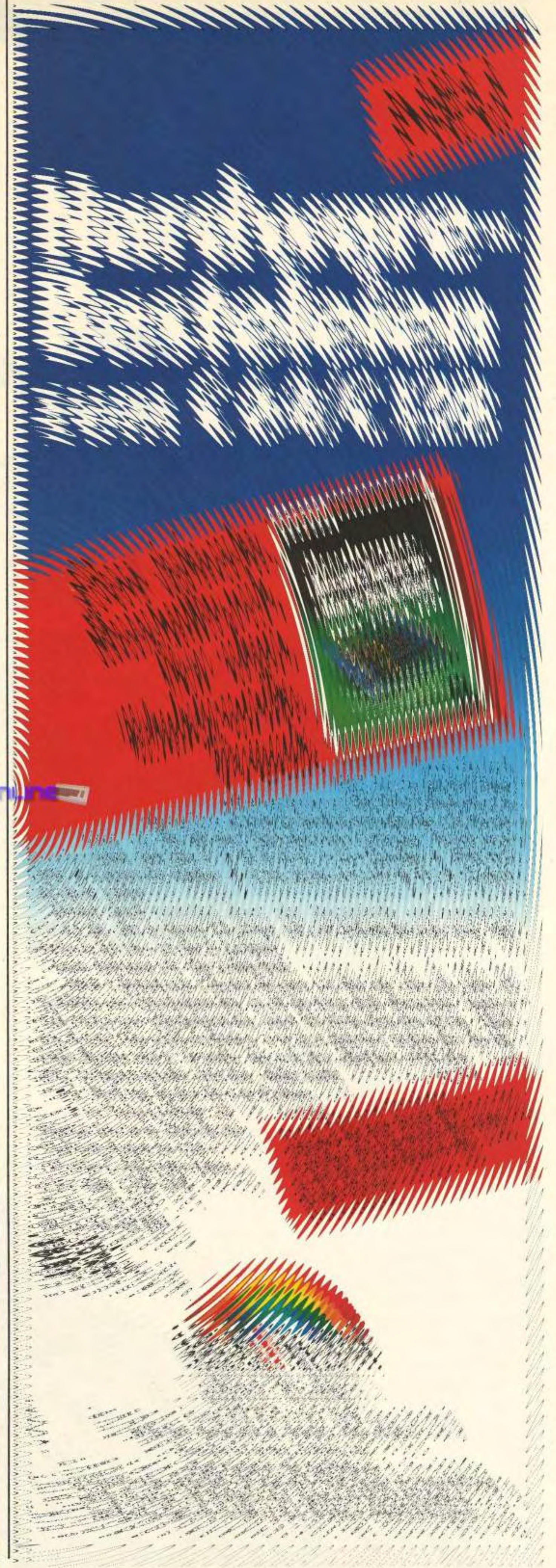






```
2349 : c5 c5 f0 0f a2 03 dd a4  12
2351 : 80 f0 0b 57 0b 2c a9 40  30
2359 : 85 cb 02 b8 13 27 e1 00  4d
2361 : 0c 13 51 4b 01 e8 bc 9c  38
2369 : 80 01 64 50 c6 88 16 61  59
2371 : 13 b9 3c c6 9d 76 02 ca  74
2379 : d0 f2 f0 db 07 92 10 7e  bc
2381 : 51 4c 1d 96 4c 49 53 54  17
2389 : 00 b0 a7 41 55 54 4f 15  53
2391 : 2e 44 55 4d 50 0d d4 53  48
2399 : 4d 41 ad 23 58 96 27 52  d2
23a1 : 55 4e 0d 1f 17 47 4f 40  ae
23a9 : 04 27 46 49 4e 44 15 04  5f
23b1 : ed b0 73 f7 e9 dc 30 a8  6a
23b9 : 41 a7 1d ad f7 a8 a4 ab  79
23c1 : be ab 80 b0 05 ac a4 a9  27
23c9 : 8a 8d 76 8d 7a 85 c5 8b  6b
23d1 : 82 a8 d1 a8 3a a9 2e a8  2c
23d9 : e2 8b 2c b8 b5 96 4a 98  0d
23e1 : b8 96 76 83 23 b8 7f aa  3e
23e9 : 09 8d 56 a8 84 99 6e 8d  4d
23f1 : 85 aa 29 e1 bd e1 c6 e1  1c
23f9 : 7a ab ae 9a 07 af 25 92  ef
2401 : 32 94 49 95 5a 8d 73 90  83
2409 : 75 8b a2 9a 13 9b 3b 9a  70
2411 : 8a 8f 13 8d 7a 87 93 8d  27
2419 : 1d 88 ae 84 d5 9c 2d 9b  e5
2421 : 21 89 17 89 fd 88 a9 9b  00
2429 : b1 84 0c 87 00 94 27 86  5f
2431 : 69 9b 20 94 3d 88 e4 9a  e3
2439 : 0c 9b 1e 91 31 85 e1 8c  ac
2441 : 74 88 fc 8f 3e 8c 11 94  e0
2449 : dd 88 07 af b9 88 31 8f  e6
2451 : 99 87 be 87 ab 84 89 86  60
2459 : 34 9b 81 88 84 f2 40 85  b8
2461 : 0e 86 60 86 6f 89 98 89  54
2469 : 38 9a 4f 46 c6 52 45 4e  3c
2471 : 55 cd 58 24 c4 44 45 cc  65
2479 : 07 48 c0 cf 71 ea d0 56  2d
2481 : 50 4c 4f d4 43 41 4c cc  6f
2489 : 42 4f 52 f7 d2 45 58 45  4a
2491 : c3 4d 45 52 47 c5 48 9c  93
2499 : 78 06 44 4f 4b c5 53 50  e0
24a1 : 41 43 c5 49 4e 50 55 54  84
24a9 : 4c 03 8a 45 d4 7b 06 e7  d3
24b1 : 43 55 52 53 4f d2 41 44  b7
24b9 : 53 d2 50 b9 76 c5 50 4c  30
24c1 : 41 d9 56 4f 4c 55 4d c5  9f
24c9 : 52 45 00 05 60 4d 45 cd  80
24d1 : 54 52 05 43 1f 2b 1b c3  37
24d9 : 20 b8 b8 4c 45 54 54 45  e0
24e1 : d2 48 45 4c d0 43 03 36  52
24e9 : 47 52 4f 55 4e c4 d0 55  d1
24f1 : da d8 44 49 53 50 4f 00  67
24f9 : 8c 27 ba d2 54 c0 48 49  21
2501 : 00 72 34 48 41 52 44 43  8f
2509 : 4f 50 d9 05 40 9e e9 b4  a1
2511 : cd 4c 4f 43 cb 53 57 41  78
2519 : d0 55 53 f6 c7 53 45 c3  fb
2521 : 45 4c 53 c5 45 52 52 4f  e9
2529 : d2 00 9c 30 0a c2 45 cb  8c
2531 : e1 f2 a3 a4 50 4f 18 43  6f
2539 : d4 03 8a d2 43 d5 a7 4d  a8
2541 : 49 ce 4d 41 d8 d5 85 50  60
2549 : 06 46 52 41 c3 4f 44 80  f8
2551 : 0c 65 c3 48 45 58 a4 45  3e
2559 : 60 cc 00 a0 01 4c d7 bd  81
2561 : ea 11 00 ff 00 ff 00 ff  d4
```

Listing 1. »EXBASIC L II« Schluß



Detektiv für Basic-Program

Ein hervorragendes Hilfsprogramm prüft Ihre Basic-Programme. Fehlerhafte Zeilen werden angezeigt und auf Wunsch ausgedruckt. Die zeitraubende und schweißtreibende Fehlersuche in einem Programm wird damit kinderleicht.

Das Basic-Kontroll-System (BKS) wurde ursprünglich entwickelt, um Leserlistings für das 64'er-Magazin auf Lauffähigkeit zu testen. Das kurze Maschinenprogramm (Listing 1) prüft ein im Speicher stehendes Basic-Programm Zeile für Zeile darauf, ob typische Fehler enthalten sind, beispielsweise: eine Klammer zu viel oder zu wenig, Syntaxfehler, Formatfehler im Programmtext. Ganz besonders hat es das Fahndungsprogramm auf Sprungbefehle abgesehen. Hier werden eigentlich alle möglichen Fehler gefunden. Doch das BKS ist sicher nicht nur für professionelle Programmtester interessant, jeder Programmierer kann durchaus etwas damit anfangen. Beispielsweise ein Einsteiger, der gerne überprüfen möchte, ob das Basic-Programm, das er gerade geschrieben hat, auch sicher läuft. Aber auch Fortgeschrittene profitieren von diesem Programm. Nehmen wir einmal an, Sie haben ein längeres (vielleicht dazu noch fremdes) Basic-Programm vorliegen, das Sie nach eigenen Vorstellungen verbessern möchten. Sie nehmen hier und da eine Zeile heraus, fügen dort eine neue ein und vertauschen einige Programmteile. Doch was ist, wenn Sie dabei irgendwo einen Befehl »GOTO XX« übersehen haben, der jetzt auf eine nicht mehr existierende Zeile weist? Das Programm wird beim Testlauf aussteigen, üblicherweise dann, wenn man es am wenigsten braucht.

Wenn Sie bereits Programmierprofi sind, kennen Sie sicher folgende Situation: Sie haben ein langes und komplexes Basic-Programm geschrieben, das von Maschinenprogrammen unterstützt wird. Da wird dann eifrig im Speicher herumgePOKEt, der Programmierer wirft mit SYS-Befehlen um sich, ein Finger hat seinen Stammplatz am RESET-Taster. Es ist bereits 2 Uhr nachts. Hoppla! Da haben Sie in der Eile einen falschen POKE-Befehl eingegeben, der das Basic-Programm angegriffen hat. Oder durch eine ungeschickte Manipulation wurde der Basic-Endezeiger (45/46) so verändert, daß ein Start des Programms den totalen Absturz zur Folge hat.


me



Magazin im Überblick

Diese 64´er- Ausgaben bekommen Sie noch bei Markt & Technik für jeweils 6,50 DM.

Tragen Sie die Nummer der gewünschten Ausgabe (z.B. 3/88) in den Bestellabschnitt der Zahlkarte nach Seite 34 ein.

8/86: Übersicht: Programmiersprachen für C64 und C128 / C- Compiler im Vergleich / Lernsoftware: C64-Programme auf einen Blick

9/86: Entscheidungshilfe: So finde ich den richtigen Drucker / Kopierschutz: Die neuen Trends / Test: Zwei Top- Assembler im Vergleich

10/86: Listing des Monats: "Der Soundmonitor" / DFÜ: Die interessantesten Mailboxen Großer Einsteiger- Sonderteil

11/86: Listing: "Spellchecker" für Vizawrite Animation: 3D- Grafik in Echtzeit Eingabegeräte: Maus und Joystick im Vergleich

12/86: Übersicht: Hardware- Erweiterungen Bauanleitung: Centronics- Interface Listing des Monats: Floppy-Speeder "Exos V3"

1/87: Spiele: Die Renner ´86, Billigspiele im Test Farbmonitore im Vergleich / Großer Einsteigerteil: So fängt man an

3/87: Zum Abtippen: Kopierprogramm der Spitzenklasse / Disketten: Markenqualität gegen No-Name- Produkte C128 : Speichererweiterungen im Test

4/87: Programmiersprachen: So arbeiten Profis Listing des Monats: Terminalprogramm "Proterm V6" Test: Farbfernsehgeräte als Monitorersatz

5/87: Fractals: Die Welt der Apfelmännchen Kaufhilfe: Die besten Floppy- Speeder 3½-Zoll- Floppy für den C64

2/88: Desktop Publishing live: Zeitung machen mit dem C64 / Tolles Malprogramm zum Abtippen

3/88: Brennpunkt Spiele: Spiele per Telefon u. a. Kopierprogramme im Vergleich

4/88: Gibt es einen neuen C64 ? / Alles über Btx und Datenfernübertragung / Große Checkliste zum Kauf von Software

5/88: C64 kontra Amiga, Atari & Co. Vergleichstest: Drucker / Im Härtetest: Neuer Super-Joystick / Großer Einsteiger- Sonderteil

6/88: Keyboards am C64 / Markendisketten im HärtetestTest: Floppy- Speeder Neuer Kurs: Assembler

8/88: Tips und Tricks zu Druckern / Basic- Kurs für Einsteiger / Alles über RAM, ROM, EPROM & Co.

9/88: Neuer Kurs: Drucker professionell nutzen Messen, Steuern, Regeln: Profigeräte im Test / Public Domain- Spiele

10/88: Test: Modems und Akustikkoppler Listing des Monats: Super- Strategie- Spiel Musikhardware im Vergleich

11/88: Publish C64: Professionelles Druckprogramm zum Abtippen / Test: Malprogramm Giga- Paint Ratgeber Druckerkauf

12/88: Weihnachts- Special: Die besten Geschenkideen / Geheimtip: Monitor für 40,-DM / Bauanleitung: Drucker- Interface

1/89: Die besten Druckprogramme / 20 Zeiler zum Abtippen / Malprogramme für den C128 im Vergleich Jahresinhaltsverzeichnis

2/89: Test: Schnellster Basic- Compiler Listing: "Master Copy Plus" / Spiele ´88 Computerschreibtisch zum Spartarif

3/89: Kaufhilfe: Floppies, Drucker, Monitore Bauanleitung: 256 KByte Zusatzspeicher / Software Test: Geos 2.0 ist da / Viren im C64

Mit diesen Sammelboxen sind Ihre Ausgaben immer sortiert und griffbereit.

Eine Sammelbox faßt einen vollständigen Jahrgang mit 12 Ausgaben und kostet 14,– DM.





Sonderhefte im Überblick

Die 64er Sonderhefte bieten Ihnen umfassende Informationen in komprimierter Form zu speziellen Themen rund um die Commodore C 64, C 128, C 16/116, VC 20 und denPlus/4. Diese Ausgaben hat Ihr Händler vorrätig - oder er bestellt sie gerne für Sie.



DRUCKER



SH 9904: GRAFIK & DRUCKER
80- Zeichen- Karte zum Abtippen / Hardcopy- Routinen für viele Drucker



SH 0018: DRUCKER
Listing: professionelle Textverarbeitung für den MPS 801 / Matrixdrucker im Test



SH 0032: FLOPPYLAUFWERKE UND DRUCKER
Tips&Tools / RAM- Erweiterung des C64 / Druckerroutinen

HARDWARE

FLOPPY, DATASETTE



SH 0013: HARDWARE
Ein- Chip- Microcomputer / Bauanleitungen: MIDI-Interface, Speicheroszilloskop, IC- Tester



SH 9905: FLOPPY / DATASETTE
Disketten kopieren mit Hypra-Copy/ 10mal schneller laden mit Turbo Tape de Luxe



SH 0009: FLOPPY / DATEIVERWALTUNG
Floppy- Beschleuniger im Vergleichstest / Arbeiten mit dBase II / C 128- Diskmonitor



SH 0015: FLOPPY / DATASETTE
Reparaturanleitung: Erste Hilfe für die Diskettenstation / Hypratape: das Super- Turbotape



SH 0025: FLOPPY- LAUFWERKE
Wertvolle Tips und Informationen für Einsteiger und Fortgeschrittene



SH 0028: GEOS / DATEIVERWALTUNG
Viele Kurse zu GEOS / Tolle GEOS-Programme zum Abtippen

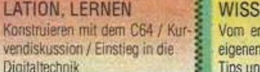
GRAFIK, SOUND


SH 0011: GRAFIK, MUSIK, ANWENDUNGEN
50 Seiten Musikprogrammierung / Vielseitige Businessgrafik

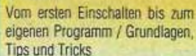

SH 0020: GRAFIK
Grafik- Programmierung / Bewegungen


SH 0023: GRAFIK, ANWENDUNGEN
Außergewöhnliche Anwendungen auf dem C 64 zum Abtippen


SH 0027: GRAFIK
AMICA Paint: Malprogramm



SH 0034: GRAFIK, SIMULATION, LERNEN
Konstruieren mit dem C64 / Kurvendiskussion / Einstieg in die Digitaltechnik

C 64 - EINSTIEG



SH 0005: C 64- GRUNDWISSEN
Vom ersten Einschalten bis zum eigenen Programm / Grundlagen, Tips und Tricks


SH 0016: EINSTEIGER 2
Spriteanimation: Zeichentrickfilm mit dem Computer / GEOS, die neue Benutzeroberfläche


SH 0019: EINSTEIGER 3
Basic- Kurs / Programm- Übersicht


SH 0026: RUND UM DEN C 64
Der C 64 verständlich für Alle mit ausführlichen Kursen

C 128


SH 0001: C 128
Das können C 128 und C 128 D / Vergleich: C 128- C 64 / die passende Peripherie


SH 0010: C 128 II
Die Geheimnisse von CP/M / Kompletter C 128- Schaltplan / Grafik für Einsteiger


SH 0022: C 128 III
Farbiges Scrolling im 80- Zeichen-Modus / 8- Sekunden- Kopierprogramm


SH 0029: C 128
Starke Software für C 128/C 128 D/ Alles über den neuen C 128 D im Blechgehäuse


SH 0036: C 128
Power 128: Directory komfortabel organisieren / Haushaltsbuch: Finanzen im Griff / 3D- Landschaften aus dem Computer

C 16/116, VC 20, PLUS/4


SH 0003: C 16/116, VC 20, PLUS/4
Listings für Spiele, Grafik, Tips &Tricks / Anwendungen: Dateiverwaltung, VC 20 mit Musik


SH 0008: PLUS/4 UND C16
Übersicht: Zeropage und wichtige Systemadressen / Grundlagen und viele Listings

SPIELE


SH 9902: ABENTEUER-SPIELE
45 Seiten Adventure- Programmierkurs / Listings und Schritt-für-Schritt-Lösungen



SH 9903: SPIELE
Top- Spiele- Listings für C 64 und VC 20 / Große Spiele- Marktübersicht


SH 0004: ABENTEUER-SPIELE
Kurs: Programmierung von Grafik, Parser und künstlicher Intelligenz / Viele Adventures


SH 0017: SPIELE FÜR C64 UND C 128
So programmiert man Scrolling / Strategiespiele: Grips ist gefragt


SH 0030: SPIELE FÜR C64 UND C 128
Tolle Spiele zum Abtippen für C 64/ C 128 / Spieleprogrammierung

TIPS&TRICKS, ANWENDUNGEN


SH 9901: TIPS&TRICKS
Befehlserweiterungen für Betriebssystem und Floppy / Unentbehrliche Programmierhilfen



SH 9906: AUSGEWÄHLTE SUPERLISTINGS
Die besten Programme aus den 64er- Magazinen 1984/85


SH 9907: ANWENDUNGEN/ DFÜ
Terminal und Mailboxprogramm zum Abtippen / Der C 64 als Winzer


SH 0002: TIPS&TRICKS
Zeichensatz- und Sprite- Editor / Interrupt- Joystickabfrage / 27 nützliche Einzeiler


SH 0024: TIPS, TRICKS & TOOLS
Die besten Peeks und Pokes sowie Utilities mit Pfiff


SH 0031: DFÜ, MUSIK, MESSEN - STEUERN - REGELN
Alles über DFÜ / BTX von A-Z / Grundlagen / Bauanleitungen


SH 0033: TIPS,TRICKS & TOOLS
Basic- Control- System / Titelgenerator / Digitale Super- Sounds / Betriebssysteme im Vergleich

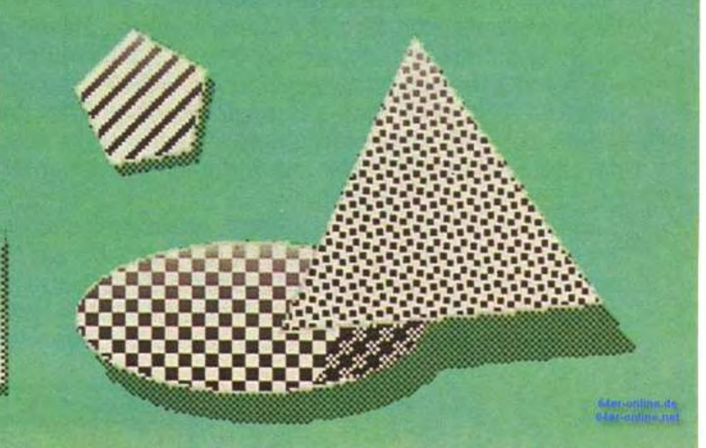
PROGRAMMIER- UND MASCHINENSPRACHE


SH 0007: PEEKS&POKES
"Maschinen- Power" mit Basic / Multitasking: 2 Basic- Programme laufen nebeneinander / Peeks und Pokes zum C 128


SH 0012: PROGRAMMIERSPRACHEN
Pascal, Comal, Prolog, C und Forth/ Vergleich: Basic- Compiler


SH 0021: ASSEMBLER UND BASIC
Giga- Ass: Hypra- Ass hoch 2 / Paradoxon- Basic: 50000 Basic Bytes free


SH 0035: ASSEMBLER
Abgeschlossene Kurse für Anfänger und Fortgeschrittene



In solch einem Fall lädt man einen Monitor und versucht, zu retten, was noch zu retten ist. Doch stimmen alle Zeiger, Zeilennummern und Sprünge jetzt noch?

In allen diesen Fällen hilft das BKS. Laden Sie die Testroutine mit »LOAD "BKS 5.0 (49152)",8,1«. Danach geben Sie bitte »NEW« ein, um alle Zeiger richtig zu stellen. Keine Angst, das Maschinenprogramm wird dabei nicht gelöscht. Es kann jederzeit mit »SYS 49152« gestartet werden. Das Programm funktioniert natürlich nur, wenn sich ein Basic-Programm im Speicher befindet. Andernfalls wird eine entsprechende Meldung ausgegeben.

Laden Sie nun also das zu testende Basic-Programm. Der Inhalt des Basic-Anfangszeigers (43/44) spielt keine Rolle. Starten Sie das BKS mit »SYS 49152«. Es erscheinen jetzt einige Fragen, die Sie mit <J> (Ja) oder <N> beantworten. Zunächst werden Sie gefragt, ob Leerzeichen bemängelt werden sollen. Manche Programmierer setzen in Basic-Programme viele Leerzeichen (Spaces), um sie übersichtlicher zu gestalten, andere verschmähen dies. Sollten Sie zu letzteren gehören, können Sie die Frage mit <J> beantworten; dann wird auch getestet, ob außerhalb von Anführungszeichen und DATAs überflüssige Spaces auftreten. Wenn in einem Programm allerdings zu viele Leerzeichen sind, besteht die Liste, die das BKS erzeugt, praktisch nur noch aus diesem Fehler und wird unübersichtlich. Schalten Sie die Überprüfung dann besser aus.

Die nächste Frage hat eine ähnliche Funktion. Das BKS kann auf Wunsch Sprünge (RUN, GOTO, GOSUB, THEN) finden, die auf eine REM-Zeile oder eine Trennzeile weisen, das ist eine Zeile dieser Art:

```
10 :
```

Das kann problematisch werden, wenn das Programm mit einem »unintelligenten« REM-Killer bearbeitet wird. Gerade in einer Zeitschrift veröffentlichte Listings sollten solche Sprünge nicht enthalten, da der eine oder andere Leser die REM-Zeilen beim Abtippen wegläßt und nicht alle Sprünge findet, die zu korrigieren sind.

Auch diese Fehlermeldung kann man »ausblenden«, wenn der Fehler allzu oft auftritt.

Mit der folgenden Frage kann man gleich zwei Fehler auf einmal ausblenden: den Fehler Nummer 7, der angezeigt wird, wenn ein Sprungbefehl auf einen Strukturierbefehl zeigt, den man auch anstelle des Sprungbefehls hätte setzen können. Beispiel:

```
10 A = 4 : GOTO 20
20 RETURN
```

kann man auch einfacher so schreiben:

```
10 A = 4 : RETURN
```

Der zweite Fehler, der von diesem »Schalter« betroffen ist, ist der Fehler Nummer 17. Er wird gemeldet, wenn der Programmierer direkt hinter das Befehlswort »THEN« einen GOTO-Befehl gesetzt hat (anstelle »THEN GOTO 12« schreibt man kürzer »THEN 12«). Da beides Schönheitsfehler sind, kann man deren Ausgabe zusammen ausblenden, indem die Frage mit <J> beantwortet wird.

Die letzte der vier »Ausblendfragen« betrifft gleich drei Fehler: Wie Sie aus der Liste ersehen können, betreffen die Fehlernummern 12, 13 und 14 Befehlswörter, die ein geübter Programmierer nicht verwendet: LET, NEW und STOP. »LET« kann man sich sowieso sparen, »NEW« und »STOP« gehören nicht in ein gutes Basic-Programm. Falls der Bediener des BKS anderer Meinung ist, kann er die Ausgabe der drei Fehlermeldungen unterdrücken, indem er die Frage mit <N> beantwortet.

Als nächstes werden Sie gefragt, ob die fehlerhaften Zeilen gelistet werden sollen oder ob die Ausgabe der Zeilennummer genügt. Beantworten Sie auch diese Frage wieder mit <J> oder <N>. Die letzte Frage dient dazu, festzulegen, auf welches Gerät die Fehlerliste ausgegeben werden soll. Drücken Sie eine der Tasten <D>, <S> oder <F>:

S: Die Fehlerliste wird auf dem Bildschirm ausgegeben. Dieser Modus ist vor allem bei sehr kurzen Programmen angebracht und zum Test, ob die ausblendbaren Fehler zu oft vorkommen.

Fehler-Nr.	Bedeutung
1 (1)	Direkt nach der Zeilennummer folgt ein Nullbyte (dies wird zum Listschutz verwendet).
2 (1)	Im Programmtext kommt ein überflüssiges Leerzeichen vor. Nach DATA, in Anführungszeichen oder wenn diese Funktion abgeschaltet ist, werden Spaces nicht bemängelt.
3 (2)	Ein Befehl THEN, GOTO, LIST, RUN oder GOSUB zeigt auf eine nicht existierende Zeile (UNDEF'D STATEMENT).
4 (2)	Die hinter einem dieser Befehle stehende Zeilennummer ist größer als 63999.
5 (2)	Die hinter einem dieser Befehle stehende Zeilennummer enthält verbotene Zeichen (etwa »GOTO 4+6« oder »GOTO LABEL«). Gerade Pascal-gewohnte Programmierer werden diese Funktion des BKS zu schätzen wissen...
6 (2)	Eine Basic-Zeile ist länger als 255 Zeichen.
7 (1)	Ein GOTO- oder THEN-Befehl zeigt beispielsweise auf RETURN, GOTO oder RUN (Strukturierbefehl), den man einfach anstelle des Sprungbefehls hätte setzen können.
8 (1)	Ein Sprungbefehl zeigt auf eine REM- oder Trennzeile (das ist eine Zeile, die nur einen Doppelpunkt enthält). Das kann zu Problemen beim Abtippen führen, wenn die REM-Zeile weggelassen wird. Falls in einem Listing dieser Fehler zu oft vorkommt, kann auch die Ausgabe dieses Fehlers abgeschaltet werden.
9 (2)	Eine Basic-Zeile ist länger als 255 Zeichen (Fehler der Hauptroutine). Wenn dieser Fehler auftritt, dürfen eventuelle sonstige Fehler nicht mehr unbedingt ernstgenommen werden, da dann das System vollkommen durcheinandergerät.
10 (2)	Ein GOTO- oder RUN-Befehl zeigt auf sich selbst (nicht hinter THEN). Beispiel: `10 GOTO 10`
11 (2)	Der Befehl CONT darf nicht im Programmtext vorkommen.
12 (1)	Der Befehl STOP sollte nicht im Programmtext vorkommen.
13 (1)	Der Befehl NEW sollte nicht im Programmtext vorkommen.
14 (1)	Der Befehl LET sollte nicht im Programmtext vorkommen.
15 (1)	Hinter einem REM-Befehl steht ein geshiftetes »L« (Listschutz).
16 (2)	Ein illegales Token (Zeichen, Byte) kommt im Programmtext vor.
17 (1)	Der Befehl GOTO sollte nicht direkt hinter THEN stehen, einer von beiden genügt.
18 (2)	Hinter einem Befehl fehlt der Parameter (z.B. OPEN).
19 (2)	Hinter GO fehlt TO.
20 (1)	Hinter GOTO, RUN usw. folgen weitere Befehle, die niemals ausgeführt werden (z.B. »GOTO 20:PRINT "GEISTERBAHN"«)
21 (2)	Klammer(n) zu viel bzw. zu wenig.
22 (1)	Das Zeichen »↑« zur Potenzierung sollte vermieden werden (große Rechenungenauigkeit).
23 (2)	Der Befehl PRINT# wurde mit ?# abgekürzt (SYNTAX ERROR).
24 (2)	Falsche Reihenfolge der Basic-Zeilen. Das kann zu Problemen bei Sprungbefehlen führen.
25 (2)	Ein falscher Linkpointer kommt im Programmtext vor. (Vor jeder Basic-Zeile steht im Speicher ein Zeiger, der angibt, wo im Speicher die nächste Zeile beginnt. Anhand dieser Zeiger »hangeln« sich unter anderem die Sprungbefehle zur gesuchten Zeile.)
26 (2)	ON ohne legalen Sprungbefehl.
27 (2)	THEN ohne IF.
28 (1)	Der Pointer 45/46 zeigt nicht genau auf das Byte hinter dem Basic-Programm. Wahrscheinlich ist noch ein Maschinenprogramm angehängt, oder es wurde ein fehlerhafter RENEW-Befehl verwendet.

Tabelle 1. Die Fehlernummern des BKS und ihre Bedeutung (in Klammern der Härtecode, Parameter C)

D: Die Liste wird auf dem Drucker ausgegeben. Die Geräteadresse ist 4, die Sekundäradresse 0. Die Routine wurde für Commodore-kompatible Drucker geschrieben, müßte jedoch auch mit anderen zusammenarbeiten. Diesen Ausgabemodus braucht man, wenn die Fehlerliste schriftlich vorliegen soll. Da dieselbe Routine wie zur Bildschirmausgabe verwendet wird, ist die Ausgabe nicht breiter als 40 Zeichen.

F: Ausgabe erfolgt auf Diskette. Dabei wird auf der (eingelegten) Floppy ein Basic-Programm mit dem Namen erzeugt, den Sie nun eingeben. Er beginnt gewöhnlich mit der Kennung »DOC.« für DOCument, diese kann durch Eingabe von <CRSR-links> jedoch überschrieben werden (Vorsicht: Den Cursor nicht weiter als auf das »D« bewegen!). Dieses File kann dann wie ein normales Basic-Programm geladen werden, ansehen können Sie sich die Auswertung durch Eingabe von LIST oder RUN (frei nach Wahl). Nach der Eingabe dieser Parameter geht das BKS das Basic-Programm nun Zeile für Zeile durch und überprüft, ob es Fehler enthält. Bei jeder neuen Zeile wechselt der Bildschirmrahmen seine Farbe. So erkennen Sie, wenn die Überprüfung läuft, daß die Routine nicht abgestürzt ist (das BKS ist so programmiert, daß es nicht abstürzen kann). Am Ende des Tests werden alle Dateien geschlossen und das BKS beendet.

Die Fehleranzeige des BKS

Jedesmal, wenn ein Fehler auftritt, wird eine Zeile mit folgendem Format ausgegeben (auf Schirm, Drucker oder Diskette):

```
AAA: BB[C] blablablabla
```

AAA ist die Basic-Zeilennummer, in der der Fehler auftritt. Sie wird rechtsbündig ausgegeben. BB ist die laufende Nummer des Fehlers (siehe dazu die untenstehende Tabelle 1).

Es gibt zwei verschiedene Fehlergrade, die »C« angibt: leichte Fehler, die eigentlich nur Schönheitsfehler sind und keine Fehlfunktion des Programms bewirken (C=1). Beispiele für diese Fehler sind überflüssige Spaces, Listschutz, LET-Befehl.

Anders ist es bei den schweren Fehlern (C=2): Hier wird sich das Programm mit einer Basic-Fehlermeldung verabschieden. Beispiele: Sprungbefehle, die auf nicht vorhandene Zeilen zeigen, formal zerstörtes Programm (POKEs) oder Syntaxfehler.

Der hier mit »blablablabla« bezeichnete Teil stellt einen Text dar, der ungefähr die Art des Fehlers angibt. Die möglichen Texte sind: »FORMATFEHLER«, »SPRUNGFEHLER«, »ÜBERFLÜSSIGER BEFEHL« und »UNERLAUBTER BEFEHL«.

Da dieser Text nicht genau die Art des Fehlers angibt, ist der Parameter BB besonders wichtig: Er kann 28 verschiedene Werte annehmen. Tabelle 1 gibt eine Übersicht über die Fehlernummern.

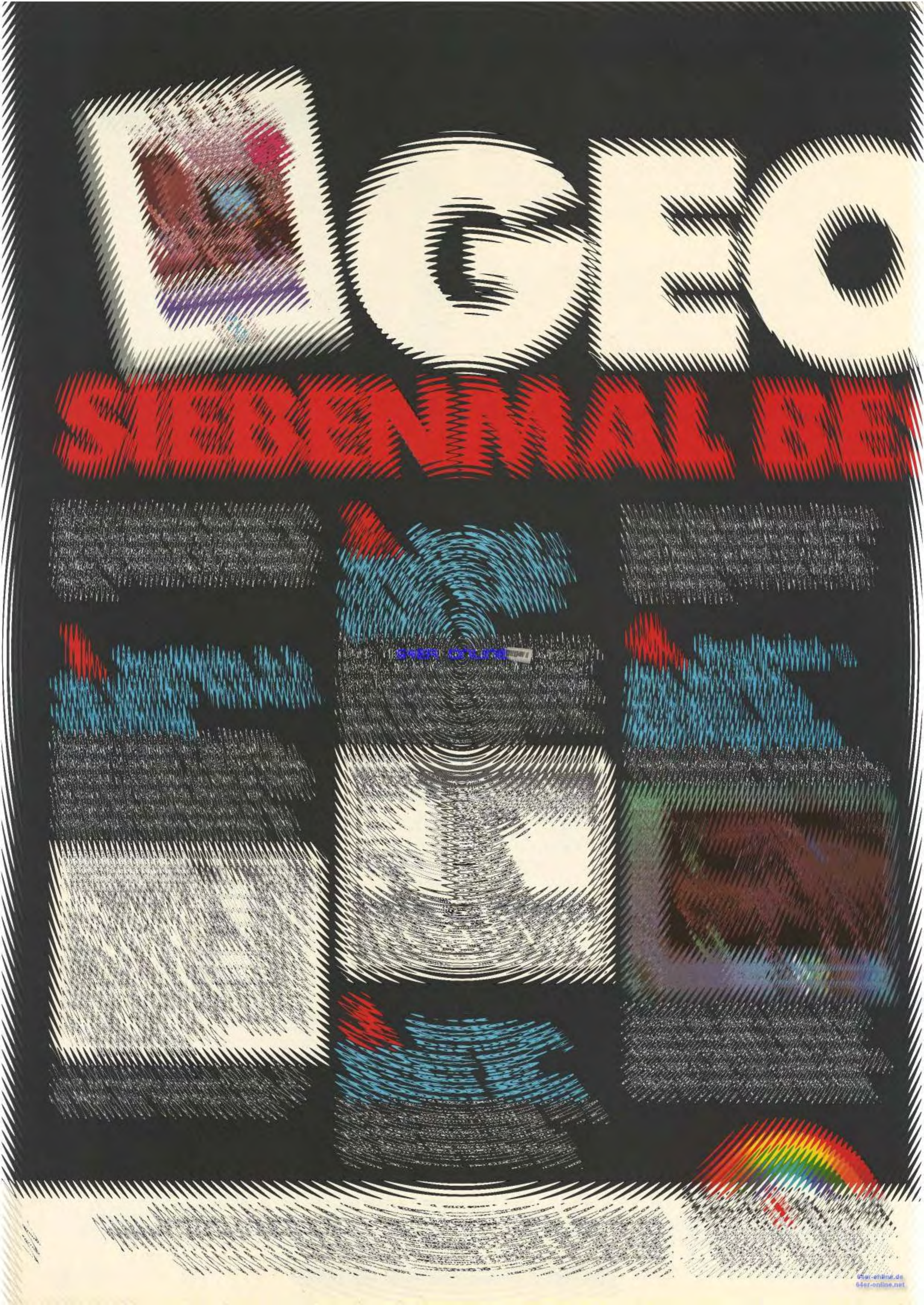
So funktioniert das BKS

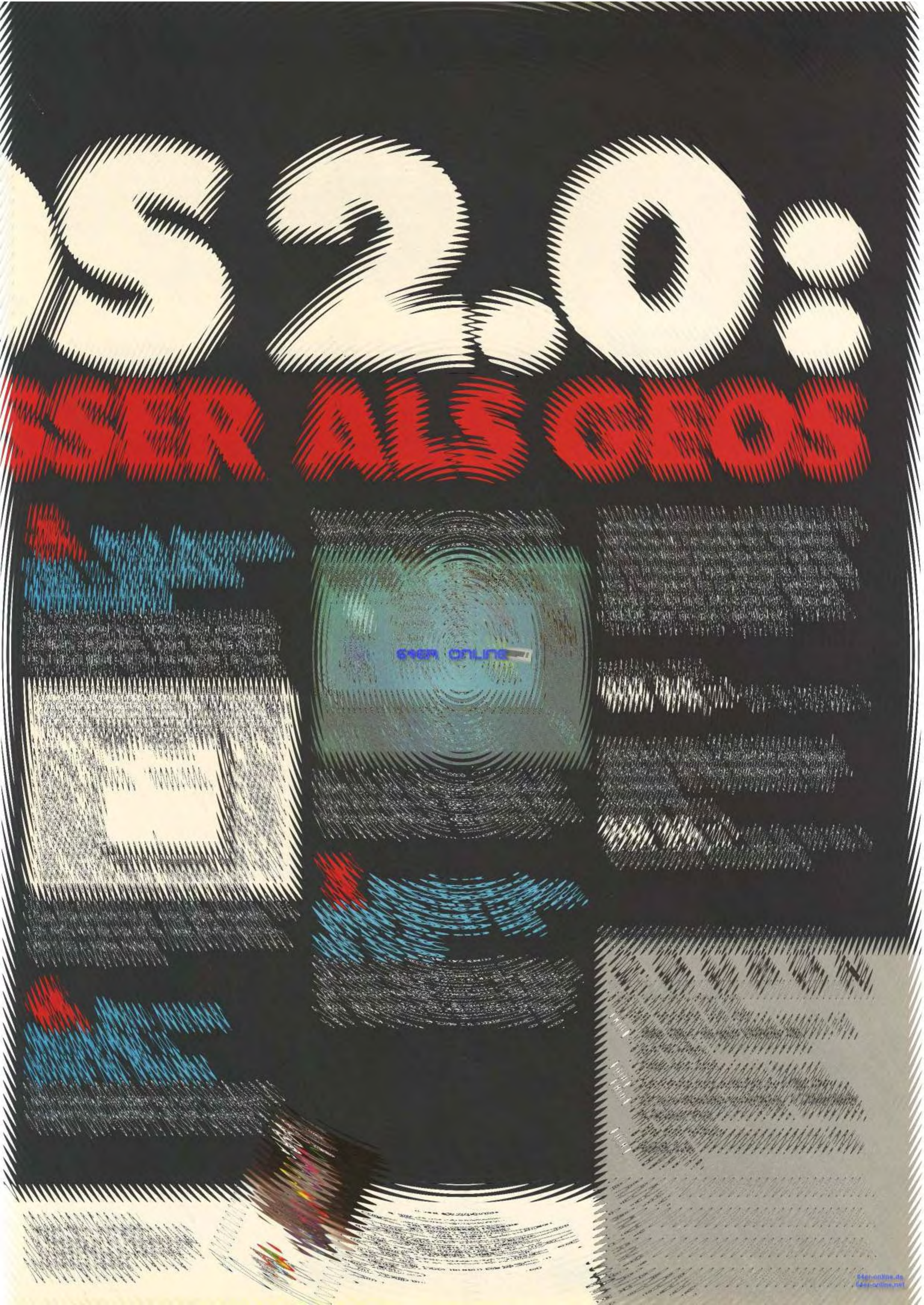
Viele Leser werden sich jetzt die Frage stellen, wie das BKS funktioniert. Zu einer detaillierten Beschreibung reicht hier leider der Platz nicht. Der Autor (Adresse im Basic-Programm, das unten beschrieben wird) stellt auf Wunsch (Einsenden einer Diskette mit Rückporto) jedoch gerne den Quelltext zur Verfügung. Daher nur ein grober Überblick:

Das Kontrollsystem geht das Basic-Programm Zeile für Zeile und Zeichen für Zeichen durch. Für die wichtigsten

Stelle	Inhalt
0002-0003	Temporär (u. a. Zeiger auf Programm)
0334-0344	Filename
C000-CCDD	Programm »BKS«
C000-C002	Sprung nach $C687 (Anfang)
C003-C239	Verschiedene Texte (gepackt)
C23A-C23B	Nummer der gesuchten Basic-Zeile für $C37C
C23C	Flag
C23D	Flag: Anführungszeichenmodus
C23E	Flag: REM
C23F-C240	Nummer der aktuellen Zeile
C241	Position in dieser Zeile
C242	Nummer des Fehlers
C243-C246	Integerzahlen für Multiplikation mit 10
C247	Aktuelles Token
C248	Aktuelle Zeigerposition
C249	Anzahl der Klammerebenen
C24A	Anzahl der Ziffern hinter Sprungbefehl
C24B	Flag: THEN
C24C	Flag: ON
C24D	Geräteadresse für Ausgabe (3, 4, 8)
C24E	Low-Byte der Anzahl der leichten Fehler
C24F	Low-Byte der Anzahl der schweren Fehler
C250	High-Byte der Anzahl der leichten Fehler
C251	High-Byte der Anzahl der schweren Fehler
C252	Flag: DATA
C253	Letzter Zustand von $C252
C254	Sekundäradresse für Ausgabe
C255	Hochkommaflag für Listroutine
C256	Flag: Basic-Zeilen listen
C257	Pointer für Listroutine
C258	»First«-Zeiger
C259-C25A	Letzte Fehlerzeilennummer
C25B	Temporär
C25C	Wie $C248
C25D	Flag: ON/Hochkomma
C25E	Wie $C248
Stelle	**Inhalt**
C25F	Letztes Zeichen
C260-C27D	Tokentabellen
C27E-C2C1	Texte der Fehler
C2C2-C2F9	Zeiger auf diese Texte
C2FA-C315	Härtecodes der Fehler (ASCII)
C316-C37B	Vorspann für erzeugtes Basic-Programm
C37C-CCDD	100% Maschinenprogramm
	Interessante Routinen:
C37C	Sucht die Zeile (X/Y)
C4DE	Entschlüsselt Text ab (A/Y) und gibt ihn aus
C562	Holt nächstes Zeichen aus Basic-Text
C56F	Holt Parameter
C584	Holt Tasten J/N
C687	Anfang des Hauptprogramms
C6D9	Schleife: Neue Basic-Zeile
C710	Hauptschleife: Nächstes Zeichen
C77A	Nächstes Zeichen
CA0D	REM-Routine
CA19	Unterroutinen zu $CACE
CACE	Gibt Fehlermeldung Nr. X aus
CBE0	fertig
CC5D	Holt Zahl hinter Sprungbefehl
CE00-CEFF	Puffer für Listroutine
CF00-CFFF	»Common-Bereich« (hier werden die Ergebnisse des BKS an »BKS.WHAT« übergeben)
CF00-CF04	Erkennungstext »NSS88«
CF05	Versionsnummer primär (=5)
CF06	Versionsnummer sekundär (=0)
CF07	Flag: Fehler Nr. 2 ausgekoppelt (0=»ja«)
CF08	Flag: Fehler Nr. 8 ausgekoppelt (0=»ja«)
CF09	Identifikationsbyte (123 = BKS läuft, 222 = BKS fertig)
CF0A	Flag: Fehler Nr. 7/17 ausgekoppelt (0=»ja«)
CF0B	Flag: Fehler Nr. 12-14 ausgekoppelt (0=»ja«)
CF0C-CF27	Tabelle aller 28 Fehler (1=»tritt auf«)

Tabelle 2. Speicherbelegung des BKS (Version 5.0, hexadezimal). Programmier-Profis werden diese Belegung verwenden, um das BKS für Ihre Zwecke mit eigenen Zusätzen auszustatten.


GEO
SIEBENMAL BE


S 2.0:
SSER ALS GEOS
64ER ONLINE

Befehle ist eine Routine vorhanden, die diese dann auf Fehlerhaftigkeit prüft. Im Falle eines Fehlers wird in eine Diagnoseroutine gesprungen, die zunächst prüft, ob der Fehler ausgeblendet ist oder angezeigt werden soll. Im letztgenannten Fall erfolgt die Ausgabe.

Aufgrund dieser Verfahrensweise ist es leider nicht möglich, komplexere Fehler, wie »NEXT WITHOUT FOR« oder »RETURN WITHOUT GOSUB« festzustellen, dazu wäre Künstliche Intelligenz notwendig. Aus langer Testerfahrung kann aber gesagt werden, daß sich das BKS zum Erkennen aller üblichen Fehler sehr gut eignet.

Nun noch zu einem »Zubehör« zum BKS, dem Basic-Programm »BKS.WHAT« (Listing 2). Wie Sie vielleicht schon bemerkt haben, gibt die Referenzliste, die das BKS erstellt, nicht besonders viel Auskunft über die Bedeutung von Fehlern. Es wird immer nur ein Pauschaltext ausgegeben. Daher gibt es dieses Zusatzprogramm, das die Liste kommentiert. Es wird wie folgt angewendet: Laden Sie wie anfangs beschrieben das BKS, dann das zu testende Programm. Führen Sie den Test durch, und lassen Sie sich die Referenz auf den Drucker (!) ausgeben. Danach laden Sie, ohne vorher den Computer auszuschalten, das Zusatzprogramm mit »LOAD "BKS.WHAT 5.0",8« und starten es mit RUN.

Referenzliste

Sofern vorher eine Auswertung mit dem BKS durchgeführt wurde, erscheint jetzt die Frage nach dem Datum und dem Namen des getesteten Basic-Programms. Alle anderen Angaben hat das BKS dem Kommentierprogramm bereits übergeben (im »Common-Bereich«, siehe Speicherbelegung, Tabelle 2). Nach kurzer Zeit wird jetzt ein ausführlicher Kommentar auf den Drucker ausgegeben, und zwar diesmal über die Sekundäradresse 7 (Groß-/Kleinschriftmodus). Darin enthalten ist eine kommentierte Liste ähnlich der auf Seite 29 abgedruckten, die Auskunft über die genaue Bedeutung der aufgetretenen Fehler gibt. Das Druckprogramm wurde für MPS-Drucker geschrieben und enthält einige Steuercodes dieser Drucker.

Ich wünsche Ihnen viel Erfolg mit dem BKS!

(Nikolaus Heusler/ef)

Kurzinfo: Basic-Kontroll-System

Programmart: Kontrollprogramm für Basic-Programme
Laden: LOAD "BKS 5.0 (49152)",8,1
Starten: Nach dem Laden des BKS unbedingt NEW eingeben. Anschließend laden Sie das Basic-Programm, das Sie überprüfen wollen. BKS wird anschließend mit SYS 49152 aktiviert.
Besonderheiten: Fehlermeldungen werden wahlweise auf Bildschirm, Drucker oder Diskette ausgegeben.
Zusatzprogramm: Mit dem Programm »BKS.WHAT 5.0« wird ein Kommentar der Fehlermeldungen ausgegeben. Bevor Sie dieses Programm anwenden, sollte ein Basic-Programm überprüft worden sein. Nach dem Test wird es, ohne den Computer auszuschalten, mit LOAD "BKS.WHAT 5.0",8 und mit RUN gestartet. Das Programm ist an MPS-Drucker angepaßt.
Programmautor: Nikolaus Heusler

```
Name : bks 5.0 (49152)   c000 ccde
----------------------------------
c000 : 4c 87 c6 1a 93 8e 08 05  dd
c008 : 04 12 19 2d 20 42 41 53  c1
c010 : 49 43 20 2d 20 4b 4f 4e  df
c018 : 54 52 4f 4c 4c 20 2d 20  ad
c020 : 53 59 53 54 45 4d 20 56  6b
c028 : 35 2e 30 19 0a 20 19 1e  e6
c030 : a3 19 0a 20 56 4f 4e 20  40
c038 : 4e 2e 48 45 55 53 4c 45  04
c040 : 52 20 2f 20 28 43 29 20  f4
c048 : 4e 53 53 20 31 37 31 30  0b
c050 : 38 37 19 2d 20 1a 03 42  73
c058 : 49 54 54 45 20 4c 41 44  7b
c060 : 45 4e 20 53 49 45 20 45  09
c068 : 52 53 54 20 45 49 4e 20  95
c070 : 20 42 41 53 49 43 2d 50  70
c078 : 52 4f 47 52 41 4d 4d 49  d4
c080 : 4e 20 20 44 45 4e 20 53  5d
c088 : 50 45 49 43 48 45 52 2e  8a
c090 : 20 20 44 41 4e 4e 20 4b  68
c098 : 4f 45 4e 4e 45 4e 20 53  d5
c0a0 : 49 45 20 20 44 49 45 54  e4
c0a8 : 45 53 54 52 4f 55 54 49  7a
c0b0 : 4e 45 20 4d 49 54 20 53  b1
c0b8 : 59 53 20 34 39 31 35 32  a0
c0c0 : 20 53 54 41 52 54 45 4e  41
c0c8 : 2e 1a 03 4c 4f c1 22 1a  0d
c0d0 : 46 45 48 4c 45 52 20 4e  58
c0d8 : 52 2e 20 03 20 49 4e 20  70
c0e0 : 5a 45 49 4c 45 20 03 55  c5
c0e8 : 45 42 45 52 46 4c 55 45  91
c0f0 : 53 53 49 47 45 20 53 50  6b
c0f8 : 41 43 45 53 20 4c 49 53  c7
c100 : 54 45 4e 03 1a 1a 46 45  01
c108 : 48 4c 45 52 20 23 20 38  1e
c110 : 20 28 47 4f 54 4f 20 2d  9b
c118 : 3e 20 52 45 4d 29 20 4c  db
c120 : 49 53 54 45 4e 03 20 3f  cd
c128 : 20 3c 4a 2f 4e 3e 19 05  24
c130 : 9d 03 1a 1a 41 55 53 47  b3
c138 : 41 42 45 20 41 [illegible] [illegible] 20  08
c140 : 12 53 92 43 48 49 52 4d  bc
c148 : 2c 20 12 44 92 52 55 43  29
c150 : 4b 45 52 2c 20 12 46 92  29
c158 : 4c 4f 50 50 59 20 3f 20  3e
c160 : 03 1a 46 49 4c 45 4e 41  d6
c168 : 4d 45 20 3f 20 03 44 4f  12
c170 : 43 2e 2c 50 2c 57 19 04  c9
c178 : 20 3d 3d 3d 3d 20 45 4e  b4
c180 : 44 45 20 3d 3d 3d 3d 03  cf
c188 : 19 04 20 4c 45 49 43 48  71
c190 : 54 45 20 46 45 48 4c 45  aa
c198 : 52 3a 20 03 19 04 20 53  49
c1a0 : 43 48 57 45 52 45 20 46  e2
c1a8 : 45 48 4c 45 52 3a 20 03  4b
c1b0 : 1a 1a 46 41 4c 53 43 48  8e
c1b8 : 45 20 5a 45 49 4c 45 4e  f5
c1c0 : 20 41 55 43 48 20 4c 49  88
c1c8 : 53 54 45 4e 03 1a 1a 42  4e
c1d0 : 41 53 49 43 2d 4b 4f 4e  7d
c1d8 : 54 52 4f 4c 4c 2d 53 59  e1
c1e0 : 53 54 45 4d 20 56 35 2e  3e
c1e8 : 30 20 28 43 29 20 4e 53  0e
c1f0 : 53 38 37 2f 38 38 1a 1a  f5
c1f8 : 03 4e 53 53 38 38 1a 1a  43
c200 : 46 45 48 4c 45 52 20 23  32
c208 : 31 37 20 55 4e 44 20 23  55
c210 : 37 20 4c 49 53 54 45 4e  1d
c218 : 03 1a 1a 46 45 48 4c 45  ca
c220 : 52 20 23 31 32 2c 20 23  bd
c228 : 31 33 20 55 4e 44 20 23  73
c230 : 31 34 20 4c 49 53 54 45  18
c238 : 4e 03 00 00 00 00 00 00  08
c240 : 00 00 00 00 00 00 00 00  41
c248 : 00 00 00 00 00 00 00 00  49
c250 : 00 00 00 00 00 00 00 00  51
c258 : 00 00 00 00 00 00 00 00  59
c260 : 80 89 8a 8e 90 9a 9b a2  ab
c268 : 89 8a 8d 9b a7 81 85 86  b7
c270 : 87 88 8b 91 92 94 96 97  a8
c278 : 98 9d 9e 9f a0 a1 46 4f  49
c280 : 52 4d 41 54 46 45 48 4c  9c
c288 : 45 52 03 53 50 52 55 4e  ab
c290 : 47 46 45 48 4c 45 52 03  93
c298 : 55 4e 45 52 4c 41 55 42  59
c2a0 : 54 45 52 20 42 45 46 45  21
c2a8 : 48 4c 03 55 45 42 45 52  a2
c2b0 : 46 4c 55 45 53 53 49 47  9e
c2b8 : 45 52 20 42 45 46 45 48  a3
c2c0 : 4c 03 7e c2 7e c2 8b c2  37
c2c8 : 8b c2 8b c2 7e c2 8b c2  a1
c2d0 : 8b c2 7e c2 8b c2 98 c2  6b
c2d8 : 98 c2 98 c2 ab c2 7e c2  a0
c2e0 : 7e c2 ab c2 7e c2 98 c2  e8
c2e8 : ab c2 7e c2 98 c2 7e c2  0b
c2f0 : 7e c2 7e c2 7e c2 7e c2  45
c2f8 : 7e c2 31 31 32 32 32 32  2c
c300 : 31 31 32 32 32 31 31 31  71
c308 : 31 32 31 32 32 31 32 31  bd
c310 : 32 32 32 32 32 31 0c 08  1b
c318 : 00 00 00 3a 3a 3a 3a 9b  f5
c320 : 00 30 08 00 00 22 0d 91  a3
c328 : 91 54 45 53 54 45 52 47  e6
c330 : 45 42 4e 49 53 3a 0d 2d  e9
c338 : 2d 2d 2d 2d 2d 2d 2d 2d  38
c340 : 2d 2d 2d 0d 0d 28 43 29  62
c348 : 52 45 41 54 45 44 20 42  93
c350 : 59 0d 12 4e 53 53 20 42  53
c358 : 41 53 49 43 2d 4b 4f 4e  05
c360 : 54 52 4f 4c 4c 2d 53 59  69
c368 : 53 54 45 4d 20 56 35 2e  c6
c370 : 30 20 4e 48 32 31 31 30  1f
c378 : 38 37 20 00 a5 02 48 a5  2b
c380 : 03 48 8e 3a c2 8c 3b c2  95
c388 : a5 2b a4 2c 85 02 84 03  f2
c390 : a0 01 a2 01 b1 02 f0 3f  e7
c398 : 18 a9 02 65 02 85 02 90  28
c3a0 : 02 e6 03 a0 00 b1 02 cd  1b
c3a8 : 3a c2 d0 08 c8 b1 02 cd  36
```

64'er SONDERHEFT

PROGRAMM-SERVICE

Direkt bestellen statt abtippen!

Die aktuelle Diskette zum Heft:

64'er-Sonderheft 40:

Das Basic-Paket für Programmierer und Anwender

Exbasic Level II:
Ehemals kommerziell vertrieben, macht diese Basic-Erweiterung Ihren C64 zu einem Kraftpaket: über 70 neue leistungsfähige Befehle – eine professionelle Programmierhilfe.

Basic-Kontroll-System:
Machen Sie Schluß mit Syntaxfehlern und unsauberem Programmier-Stil in Ihren Basic-Programmen! Wie ein Detektiv spürt das Basic-Kontroll-System die häufigsten Programmfehler auf und hebt so das Qualitätsniveau Ihrer Software.

Synthesizer:
Experimentieren Sie mit den Sound-Fähigkeiten Ihres C64: Die Tastatur wird zum Keyboard, mit dem Sie dem Computer phantastische Klänge entlocken.

Kreuzworträtsel:
Mit diesem hervorragenden Programm wird es ein Kinderspiel, selbst schwierigste Kreuzworträtsel zu erstellen. Die einfache Bedienung und die gelungene Druckausgabe der fertigen Rätsel bieten alle Voraussetzungen für viele Stunden Rätselspaß.

Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des 64'er-Sonderhefts 40 mit einem Diskettensymbol gekennzeichnet sind.

Diskette für C64/C128
Bestell-Nr. 15940

DM 29,90*

(sFr 24,90/öS 299,–*)*

*** Unverbindliche Preisempfehlung**



10 Leerdisketten 5¼" zum Sonderpreis von **DM 19,90**
Bestell-Nr. 39000
2seitig, doppelte Dichte DS/DD, 40 Spuren, 48 tpi mit Verstärkungsring und Schreibschutzkerbe inkl. Labelset, unformatiert.





Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8012 Haar bei München, Telefon (089) 4613-0





Sie suchen packende Spiele, hilfreiche Utilities und professionelle Anwendungen für Ihren Computer? Sie wünschen sich gute Software zu vernünftigen Preisen? Hier finden Sie beides! Unser stetig wachsendes Sortiment enthält interessante Listing-Software für alle gängigen Computertypen. Jeden Monat erweitert sich unser aktuelles Angebot um eine weitere interessante Programmsammlung für jeweils einen Computertyp.
Wenn Sie Fragen zu den Programmen in unserem Angebot haben, rufen Sie uns an:
Telefon (089) 4613-640

Bestellungen bitte *nur gegen Vorauskasse* an: Markt&Technik Verlag AG, Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, D-8013 Haar, Telefon (089) 4613-0.
Schweiz:
Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656.
Österreich:
Microcomput-ique, E. Schiller, Fasangasse 24, A-1030 Wien, Telefon (0222) 785661;
Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Telefon (0222) 833196.
Bestellungen aus anderen Ländern bitte nur schriftlich an: Markt&Technik Verlag AG, Abt. Buchvertrieb, Hans-Pinsel-Straße 2, D-8013 Haar, und gegen Bezahlung der Rechnung im voraus.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die abgedruckte Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungs-Scheck mit Ihrer Bestellung. Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

64'er-Sonderheft 39: Das Komplett-Paket für professionelles Desktop Publishing

Giga-Publish: Ideal für Vereine oder Schülerzeitungen: Giga-Publish ist ein professionelles Desktop-Publishing-Programm der Spitzenklasse. Gestalten Sie Ihre Texte und Grafiken zu einem perfekten Layout. Acht Seiten mit zehn verschiedenen Zeichensätzen können Sie gleichzeitig bearbeiten. **Mastertext:** Dieses Textverarbeitungsprogramm der Spitzenklasse bietet durch seine einfache Menüsteuerung einen hohen Bedienungskomfort. Es ist die ideale Ergänzung für Giga-Publish. **Master-Address:** Adreßverwaltung, Serienbriefe, Adreßaufkleber und Etiketten sind die Domäne von Master-Address. Es ist optimal auf Mastertext abgestimmt und bildet mit diesem ein leistungsfähiges Software-Paket. **Master-Spell:** Nie wieder Tipp- und Flüchtigkeitsfehler: Eine automatische Rechtschreibprüfung ist für alle unentbehrlich, die mit Mastertext arbeiten. **Hi-Eddi:** Dieses komfortable und leistungsfähige Zeichen- und Konstruktionsprogramm macht es einfach, Grafiken mit Joystick oder Maus zu erstellen. Mit Mastertext und Hi-Eddi verfügen Sie über das ideale Gespann, Giga-Publish als DTP-Programm optimal zu nutzen. Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des 64'er-Sonderhefts 39 mit einem Diskettensymbol gekennzeichnet sind. Diskette für C64/C128

Bestell-Nr. 15939 **DM 19,90*** sFr 17,-*/öS 199,-*

64'er-Sonderheft 38: Komplettes Einsteiger-Paket Diskette 1, Grafik total:

Eine Auswahl faszinierender Bilder, die zeigen, welche grafischen Fähigkeiten im C64 stecken. **Paint Magic:** Ein tolles Grafikprogramm, mit dem Sie in kürzester Zeit wunderschöne farbige Grafiken und Bilder auf den Monitor zaubern. **Alpha Drummer:** Dieses Schlagzeug-Programm liefert 24 perfekte Sounds. Wer möchte, kann beliebige Rhythmen erzeugen oder eigene Sounds digitalisieren. **Sounds zum Genießen:** Entlocken Sie dem Sound-Chip Ihres C64 Musik, die Sie vom Hocker reißt. Von Klassik bis Pop – für jeden ist etwas dabei.

Bestell-Nr. 15938 **DM 19,90*** sFr 17,-*/öS 199,-*

Diskette 2 Leichter lernen mit dem Computer:

Jetzt ist der Frust beim öden Pauken vorbei. Vier Programme helfen dabei: Der **»Vokabeltrainer«** bringt mehr Spaß beim Englischlernen. Ein Übungsprogramm zum **»Bruchrechnen«** erleichtert Schülern, diese gefürchtete Hürde zu überwinden. **»Lateinische Deklinationen«** greift auch bei dieser Fremdsprache unter die Arme. **»CAT«** bringt den »Kleineren« die Grundrechenarten mit grafischer Unterstützung näher. **Springvogel:** Helfen Sie dem Springvogel beim Eiersammeln: ein Spiel für geschickte Hände. Mit dem eingebauten Editor können Sie eigene Spielstufen erstellen und die Schwierigkeit Ihren Bedürfnissen anpassen. **Pro-Disk:** Mit dieser professionellen Diskettenverwaltung behalten Sie stets den Überblick über Ihre Programmsammlung.

Bestell-Nr. 16938 **DM 19,90*** sFr 17,-*/öS 199,-*

Aktionspreis: Alle Programme, die im Inhaltsverzeichnis des 64'er-Sonderheftes 38 mit einem Diskettensymbol gekennzeichnet sind (inkl. der Programme aus Diskette 1 und 2) erhalten Sie auf zwei Disketten
Zwei Disketten für C64/C128
Bestell-Nr. 17938 **DM 29,90*** sFr 24,90*/öS 299,-*

64'er-Sonderheft 36: Programme für alle C128-Besitzer

Haushaltsbuch – die Finanzen im Griff: Bringen Sie Ordnung in Ihre Finanzen mit dem Haushaltsbuch für den C128. Durch die Kontrolle über Ihre Ausgaben und gezieltes Wirtschaften, bleibt Ihnen sicher noch etwas über für »die kleinen Annehmlichkeiten des Lebens«. **Professionelle Schachturnierverwaltung:** Ein Profi-Programm, das schon bei mancher Schachmeisterschaft, so zum Beispiel den 15. Dortmunder Schachtagen, eingesetzt wurde. Bis zu 254 Spieler können nach dem »Schweizer System« verwaltet werden. **Power 128:** Ein universelles Disketten-Tool, das den Eigenheiten der Floppy 1571 angepaßt ist. Sie können das Directory beliebig manipulieren, sortieren und sich als Zugabe eine Sicherheitskopie des Directorys anlegen, die dem »Read Error 18 01« seine fatalen Folgen nimmt. **Spiel-Spaß total – Super-Vectors:** Vectors, dem Motorrad-Rennen aus dem Film »Tron« nachempfunden, erleben Sie hier in vollkommen neuen Grafik-Dimensionen. Die Bildschirmauflösung beträgt nun 736x354 Punkte. Dies ist mehr als professionelle PCs mit einer Hercules-Grafikkarte darstellen können. Speziell für den C128D im Blechgehäuse mit 64 Kbyte Video-RAM und alle nachträglich aufgerüsteten C128 wurde dieses Spiel umgeschrieben. Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des 64'er-Sonderhefts 36 mit einem Diskettensymbol gekennzeichnet sind. Diskette für C64/C128

Bestell-Nr. 15836 **DM 29,90*** sFr 24,90*/öS 299,-*

64'er-Sonderheft 34: Aufbruch in die dritte Dimension!

Fantastische Perspektiven: Ein Grafikprogramm mit ganz neuen Leistungsmerkmalen. Das Konstruieren von perspektivischen Grafiken, wie etwa ganzen Straßenzügen, wird zum Vergnügen. **Ein Freezer für harte Fälle:** Auf Knopfdruck wird der gesamte Speicher des C64 auf Diskette gespeichert und bei Bedarf wieder geladen. Sie arbeiten an der gleichen Stelle weiter, als wäre nichts geschehen. Sicherheitskopien von kopiergeschützten Originaldisketten anfertigen, Spielstände bei hartnäckigen Games »einfrieren« und bei Bedarf wieder laden – all das sind Stärken des Freezers. **3D-Movie-Maker:** Das Konstruieren von dreidimensionalen Körpern und deren fließende Bewegung am Bildschirm zeichnet dieses Programm aus. Nach der Eingabe der Koordinaten für den Körper und seine Bewegung erzeugen Sie faszinierende Trickfilme in 3-D. **Perfekte Simulation:** Selten zuvor wurde so anschaulich gezeigt, wie eine Braunsche Röhre – Urahn aller Bildschirme und Monitore – funktioniert. Per Tastendruck steuern Sie alle Parameter. Veränderungen des Elektronenstrahls werden in Echtzeit am Bildschirm angezeigt. **Kurvendiskussion perfekt:** Ein Segen für alle, die sich mit Mathematik und vor allem der Kurvendiskussion beschäftigen. Nicht nur, daß jede Funktion am Bildschirm anschaulich dargestellt wird. Auch die Ableitungen, Nullstellen etc. werden sofort berechnet und ausgegeben. **Digital einfach:** Ideal für den Einstieg in die Digital-Elektronik ist dieses Programm. Alle Grundfunktionen, wie AND-, OR-, EXOR-Gatter etc. werden am Bildschirm dargestellt und in ihren Funktionen simuliert. Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des 64'er-Sonderhefts 34 mit einem Diskettensymbol gekennzeichnet sind. Diskette für C64/C128

Bestell-Nr. 15834 **DM 29,90*** sFr 24,90*/öS 299,-*

64'er-Sonderheft 29: Programme, die jeder C128-Besitzer braucht

Mastertext 128: Die Super-Textverarbeitung für den 80-Zeichen-Modus mit eingebauter Rechtschreibprüfung. Komfort und Funktionsvielfalt werden bei diesem Programm großgeschrieben. Alle Standardbefehle der modernen Textverarbeitung, ein integrierter Taschenrechner und sogar der Datenaustausch per DFÜ sind enthalten. Als besonderen Leckerbissen bietet Mastertext 128 eine Rechtschreibprüfung, deren Wörterbuch beliebig erweiterbar ist. Tippfehler gehören damit der Vergangenheit an! **Der Hexer:** Endlich ein leistungsstarkes Kopierprogramm für den C128. Kopieren Sie nach Herzenslust, der Hexer wird auch Ihre Disketten bezaubern. Neben ganzen Disketten sind mit diesem Programm auch einzelne Files zu kopieren. Der Bedienkomfort des Hexers ist kaum zu überbieten. Probleme mit den verschiedenen Versionen des C128 kennt der Hexer nicht, es stehen verschiedene Versionen »für alle Fälle« bereit. Besitzer des »Dolphin-DOS« können sich über eine Version freuen, die mit diesem Floppy-Beschleuniger zusammenarbeitet. **Unidat Pro:** Der Wunsch jedes ernsthaften Computer-Anwenders ist eine leistungsfähige und komfortable Dateiverwaltung. Mit Unidat Pro wird dieser Wunsch Realität. Erstellen und verändern Sie eigene Datei-Masken. Hohe Zugriffsgeschwindigkeit auf Ihre Daten, die Unterstützung von Paßwörtern zum Datenschutz und eine Export-Funktion zeichnen diese Dateiverwaltung aus. Die Suche nach einem Datensatz erfolgt blitzschnell. **Mancomania:** Spielen Sie gerne Wirtschaftsspiele? Wenn Ihnen diese Spielgattung gefällt, ist Mancomania das Richtige für Sie. Das Spielziel ist allerdings ein wenig anders als bei den üblichen Vertretern dieses Genres: Verschleudern Sie Ihr Vermögen, so schnell Sie können. Vertreiben Sie sich die Zeit im Spiel-Casino, kaufen Sie Aktien an der Börse, und wetten Sie beim Autorennen. Denken Sie daran, das Geld muß weg. Aber das ist leichter gesagt als getan, als Millionär hat man's halt schwer! Diskette für C128

Bestell-Nr. 15829 **DM 29,90*** sFr 24,90*/öS 299,-*

*Unverbindliche Preisempfehlung

Übrigens: Mit den Gutscheinen aus dem »Super-Software-Scheckheft« für DM 149,– können Sie sechs Software-Disketten Ihrer Wahl aus dem Programm-Service-Angebot der Zeitschriften

PC Magazin	**Happy-Computer-Sonderheft**	**Computer persönlich**
PC Magazin Plus	**Amiga-Magazin**	**64'er-Magazin**
Happy-Computer	**Amiga-Sonderheft**	**64'er-Sonderheft**

bestellen – egal, ob diese DM 29,90 oder DM 34,90 kosten. Das Scheckheft können Sie per Verrechnungsscheck oder mit der eingehefteten Zahlkarte direkt beim Verlag bestellen. Kennwort: Software-Scheckheft, Bestell-Nr. 39100.

```
c3b0 : 3b c2 f0 42 a0 01 c8 f0  e8
c3b8 : 19 b1 02 d0 f9 c0 02 d0  94
c3c0 : 04 a0 06 d0 f1 c8 98 18  a8
c3c8 : 65 02 85 02 90 c2 e6 03  91
c3d0 : d0 be a2 02 2c a2 03 8e  e9
c3d8 : 3c c2 68 85 03 68 85 02  ce
c3e0 : ad 3a c2 cd 3f c2 d0 0d  7c
c3e8 : ad 3b c2 cd 40 c2 d0 05  04
c3f0 : a9 05 8d 3c c2 60 c8 b1  bc
c3f8 : 02 a2 04 c9 8f f0 d8 c9  fd
c400 : 3a f0 0d a2 07 dd 60 c2  b0
c408 : f0 cb ca 10 f8 e8 f0 c7  bd
c410 : c8 f0 c4 b1 02 f0 c0 c9  f6
c418 : 20 f0 f5 d0 dc c9 8b d0  34
c420 : 03 ee 4b c2 c9 ae d0 0b  31
c428 : 98 48 a2 16 20 ce ca 68  c4
c430 : a8 a9 ae c9 83 d0 03 ee  3a
c438 : 52 c2 c9 a7 d0 10 ae 4b  31
c440 : c2 d0 0b 98 48 a2 1b 20  86
c448 : ce ca 68 a8 a9 a7 60 a2  49
c450 : 02 4c c9 ff 20 d2 ff 2c  dc
c458 : a9 00 4c d2 ff ad 3f c2  5f
c460 : 8d 59 c2 ad 40 c2 8d 5a  05
c468 : c2 c8 b1 02 8d 3f c2 c8  aa
c470 : b1 02 8d 40 c2 ad 58 c2  0e
c478 : d0 05 ee 58 c2 d0 1a ad  08
c480 : 5a c2 cd 40 c2 90 12 d0  51
c488 : 08 ad 59 c2 cd 3f c2 90  18
c490 : 08 a2 18 2c a2 19 4c ce  37
c498 : ca a0 03 c8 f0 04 b1 02  86
c4a0 : d0 f9 c0 04 f0 f5 c8 98  30
c4a8 : 18 65 02 8d 5b c2 a5 03  0d
c4b0 : 69 00 a0 01 d1 02 d0 dc  8c
c4b8 : 88 ad 5b c2 d1 02 d0 d4  60
c4c0 : 60 ad 41 c2 48 20 62 c5  3a
c4c8 : c9 3a f0 f9 c9 8f f0 09  18
c4d0 : c9 83 f0 05 a2 14 20 ce  20
c4d8 : ca 68 8d 41 c2 60 aa a5  87
c4e0 : 02 48 a5 03 48 86 02 84  9a
c4e8 : 03 a0 00 b1 02 f0 33 c9  7a
c4f0 : 03 f0 2f c9 1a d0 02 a9  f4
c4f8 : 0d c9 19 d0 1a c8 b1 02  fd
c500 : aa c8 b1 02 20 d2 ff ca  e9
c508 : d0 fa 18 a9 03 65 02 85  ff
c510 : 02 90 d6 e6 03 d0 d2 20  2f
c518 : d2 ff e6 02 d0 cb e6 03  f1
c520 : d0 c7 68 85 03 68 85 02  2c
c528 : 60 48 ad 43 c2 0a 8d 45  bd
c530 : c2 ad 44 c2 2a b0 28 8d  16
c538 : 46 c2 a2 03 0e 43 c2 2e  4b
c540 : 44 c2 b0 1b ca d0 f5 ad  db
c548 : 43 c2 18 6d 45 c2 8d 43  c7
c550 : c2 ad 44 c2 6d 46 c2 b0  c8
c558 : 06 8d 44 c2 68 18 60 68  28
c560 : 38 60 ee 41 c2 ac 41 c2  c8
c568 : b1 02 c9 20 f0 f4 60 a9  1c
c570 : e7 a0 c0 20 de c4 a9 00  96
c578 : 8d 54 c2 20 84 c5 8e 07  a3
c580 : cf 4c 50 c6 a9 26 a0 c1  34
c588 : 20 de c4 20 e4 ff a2 00  25
c590 : c9 4e f0 06 c9 4a d0 f3  97
c598 : a2 01 20 d2 ff a0 03 20  6e
c5a0 : 3f ab 88 10 fa 60 a9 32  97
c5a8 : a0 c1 20 de c4 20 e4 ff  ee
c5b0 : c9 53 f0 0e c9 46 f0 07  c1
c5b8 : c9 44 d0 f1 a2 04 2c a2  56
c5c0 : 08 2c a2 03 8e 4d c2 20  86
c5c8 : d2 ff 20 d7 aa e0 08 d0  10
c5d0 : 3c a9 61 a0 c1 20 de c4  6f
c5d8 : a2 04 86 c6 8e 54 c2 bd  09
c5e0 : 6e c1 9d 77 02 ca 10 f7  2c
c5e8 : a2 00 20 cf ff c9 0d f0  f1
c5f0 : 08 9d 34 03 e8 e0 10 90  2b
c5f8 : f1 a0 00 b9 72 c1 9d 34  85
c600 : 03 e8 c8 c0 04 90 f4 8a  6f
c608 : a2 34 a0 03 2c a9 00 20  9d
c610 : bd ff ae 4d c2 a9 02 ac  fd
c618 : 54 c2 20 ba ff 20 c0 ff  31
c620 : 20 4f c4 ae 4d c2 e0 08  6d
c628 : d0 19 a9 01 20 d2 ff a9  fb
c630 : 08 20 d2 ff a2 00 bd 16  4a
c638 : c3 20 d2 ff e8 e0 66 90  10
c640 : f5 b0 07 a9 cd a0 c1 20  ad
c648 : de c4 20 cc ff 4c d7 aa  41
c650 : a9 04 a0 c1 20 de c4 20  a8
c658 : 84 c5 8e 08 cf a9 fe a0  eb
c660 : c1 20 de c4 20 84 c5 8e  dc
c668 : 0a cf a9 19 a0 c2 20 de  46
c670 : c4 20 84 c5 8e 0b cf a9  f2
c678 : b0 a0 c1 20 de c4 20 84  8a
c680 : c5 8e 56 c2 4c a6 c5 20  cc
c688 : 81 ff a9 03 a0 c0 20 de  22
c690 : c4 20 a0 cc a9 06 8d 20  67
c698 : d0 8d 21 d0 a0 01 b1 2b  c0
c6a0 : d0 14 a2 04 86 c6 bd cb  d1
c6a8 : c0 9d 77 02 ca 10 f7 a9  b5
c6b0 : 57 a0 c0 4c de c4 a9 02  d0
c6b8 : 20 c3 ff 20 6f c5 a5 2b  d0
c6c0 : 85 02 a5 2c 85 03 a9 00  4c
c6c8 : a2 03 9d 4e c2 ca 10 fa  d6
c6d0 : 8d 49 c2 8e 03 ce 8d 58  12
c6d8 : c2 ee 20 d0 ad 49 c2 f0  46
c6e0 : 0a ad 52 c2 d0 05 a2 15  98
c6e8 : 20 ce ca a0 01 b1 02 d0  7d
c6f0 : 03 4c e0 cb 20 5d c4 a9  1e
c6f8 : 00 8d 4b c2 8d 3d c2 8d  d3
c700 : 3e c2 8d 49 c2 8d 4c c2  7b
c708 : 8d 52 c2 a9 04 8d 41 c2  db
c710 : ac 41 c2 b1 02 d0 1a c0  d4
c718 : 04 d0 09 a2 01 20 ce ca  fd
c720 : a9 09 d0 e9 c8 98 18 65  3c
c728 : 02 85 02 90 ac e6 03 d0  2f
c730 : a8 c9 22 d0 10 ad 3d c2  48
c738 : 49 ff 8d 3d c2 4c 7a c7  94
c740 : a2 09 4c e7 c7 ae 3d c2  e3
c748 : d0 30 ae 3e c2 d0 33 c9  b7
c750 : a2 d0 04 a2 0d d0 20 c9  1b
c758 : 9a d0 04 a2 0b d0 18 c9  db
c760 : 20 d0 0e a2 02 ad 07 cf  0a
c768 : f0 10 ad 52 c2 d0 0b f0  d7
c770 : 06 c9 90 d0 10 a2 0c 20  20
c778 : ce ca ee 41 c2 f0 c1 4c  e2
c780 : 10 c7 4c 0d ca c9 8f d0  03
c788 : 1e ee 3e c2 d0 ec ad 47  bf
c790 : c2 c9 8a f0 0c c9 89 f0  0f
c798 : 08 c9 9b f0 04 c9 a7 d0  59
c7a0 : 51 20 c1 c4 4c f2 c7 20  c6
c7a8 : 1d c4 a2 04 dd 68 c2 f0  5e
c7b0 : 05 ca 10 f8 30 39 8d 47  cf
c7b8 : c2 20 5d cc ca f0 cf ca  84
c7c0 : f0 30 ca f0 0a ad 5c c2  9e
c7c8 : 8d 41 c2 a2 04 d0 18 c9  b5
c7d0 : fc d0 05 ad 4c c2 d0 1a  7e
c7d8 : ad 5c c2 8d 41 c2 a2 05  d4
c7e0 : ad 47 c2 c9 a7 f0 03 20  69
c7e8 : ce ca ac 41 c2 b1 02 4c  c9
c7f0 : 67 c8 ae 43 c2 ac 44 c2  f7
c7f8 : c0 fa b0 c9 20 7c c3 ad  eb
c800 : 3c c2 f0 16 c9 01 d0 15  ae
c808 : a2 03 ad 47 c2 c9 9b f0  4b
c810 : 09 c9 8a d0 38 ad 4a c2  5a
c818 : d0 33 4c 10 c7 c9 02 d0  0b
c820 : 04 a2 06 d0 28 c9 05 d0  97
c828 : 16 a2 0a ad 5c c2 8d 41  5c
c830 : c2 ad 4b c2 d0 b4 ad 47  ec
c838 : c2 c9 9b f0 ad d0 a8 c9  7b
c840 : 03 d0 12 ae 4c c2 d0 d2  c9
c848 : a2 07 2c a2 08 ad 5c c2  b2
c850 : 8d 41 c2 d0 92 ad 47 c2  82
c858 : c9 9b f0 be c9 8a d0 eb  0f
c860 : ad 4a c2 d0 e6 f0 b3 c9  55
c868 : 88 d0 05 a2 0e 4c 77 c7  9f
c870 : c9 ff f0 14 c9 cc 90 04  45
c878 : a2 10 d0 42 c9 80 b0 08  12
c880 : c9 60 b0 f4 c9 20 90 f0  06
c888 : c9 a7 d0 16 ae 41 c2 20  5c
c890 : 62 c5 8e 41 c2 c9 89 d0  e3
c898 : 04 a2 11 d0 21 ac 41 c2  4e
c8a0 : b1 02 c9 cb d0 20 ad 41  85
c8a8 : c2 8d 48 c2 20 62 c5 c9  5b
c8b0 : a4 f0 0e a2 13 2c a2 12  e5
c8b8 : ad 48 c2 8d 41 c2 4c 77  36
c8c0 : c7 a9 89 4c b6 c7 c9 cc  b2
c8c8 : b0 2c c9 80 90 28 c9 a3  c9
c8d0 : b0 0c a2 10 dd 6d c2 f0  67
c8d8 : 05 ca 10 f8 30 18 ad 41  62
c8e0 : c2 8d 48 c2 20 62 c5 c9  93
c8e8 : 3a f0 cb aa f0 c8 ac 48  7b
c8f0 : c2 8c 41 c2 b1 02 c9 28  44
c8f8 : d0 03 ee 49 c2 c9 29 d0  ef
c900 : 03 ce 49 c2 c9 3a d0 21  09
c908 : ae 52 c2 8e 53 c2 a2 00  38
c910 : 8e 4c c2 8e 52 c2 ae 49  cf
c918 : c2 f0 0e a2 00 8e 49 c2  49
c920 : ae 53 c2 d0 04 a2 15 d0  8e
c928 : 95 c9 a6 d0 03 ee 49 c2  b8
c930 : c9 a3 d0 03 ee 49 c2 c9  37
c938 : 91 d0 03 ee 4c c2 c9 99  05
c940 : d0 1a ad 41 c2 8d 48 c2  f0
c948 : 20 62 c5 ac 48 c2 8c 41  f0
c950 : c2 c9 23 d0 04 a2 17 d0  2d
c958 : ce 4c 7a c7 c9 91 d0 f9  44
c960 : a9 00 8d 5d c2 ad 41 c2  3c
c968 : 8d 5e c2 20 62 c5 aa d0  7a
c970 : 0d a2 1a 20 ce ca ad 5e  10
c978 : c2 8d 41 c2 d0 db c9 22  01
c980 : d0 0b ad 5d c2 49 ff 8d  7e
c988 : 5d c2 4c 6b c9 ae 5d c2  d4
c990 : d0 d9 c9 3a f0 db c9 89  2f
c998 : f0 04 c9 8d d0 cd 8d 47  ef
c9a0 : c2 20 5d cc ca f0 16 ca  85
c9a8 : f0 13 ca f0 04 a2 04 d0  fa
c9b0 : 06 c9 fc f0 08 a2 05 20  e2
c9b8 : ce ca 4c f7 c9 8d 5f c2  09
c9c0 : ae 43 c2 ac 44 c2 c0 fa  a9
c9c8 : b0 e3 20 7c c3 ad 3c c2  22
c9d0 : f0 25 c9 01 d0 04 a2 03  a3
c9d8 : d0 dd c9 02 d0 04 a2 06  0d
c9e0 : d0 d5 c9 05 d0 09 a2 0a  a2
c9e8 : ad 4b c2 d0 0a f0 c8 c9  e4
c9f0 : 03 f0 04 a2 08 d0 c0 ad  26
c9f8 : 5f c2 f0 0e c9 fc f0 a1  42
ca00 : ad 47 c2 c9 8d f0 03 20  e7
ca08 : c1 c4 4c 76 c9 c9 cc d0  cd
ca10 : 05 a2 0f 4c 77 c7 4c 7a  90
ca18 : c7 ae 3f c2 ad 40 c2 20  87
ca20 : 96 ca a9 3a 20 d2 ff 20  a6
ca28 : 3f ab ae 42 c2 a9 00 20  ea
ca30 : c1 ca ad 01 01 d0 10 ad  14
ca38 : 00 01 8d 01 01 a9 00 8d  b5
ca40 : 02 01 a9 20 8d 00 01 20  4e
ca48 : 0c bf 20 de c4 a9 5b 20  5f
ca50 : d2 ff ae 42 c2 bd f9 c2  9d
ca58 : 20 d2 ff 20 89 ca a9 5d  35
ca60 : 20 d2 ff 20 3f ab ae 42  7e
ca68 : c2 ca 8a 0a aa bd c3 c2  a0
ca70 : a8 bd c2 c2 20 de c4 ae  69
ca78 : 4d c2 e0 08 f0 03 20 d7  b7
ca80 : aa a9 00 20 d2 ff 4c cc  fb
```

Listing 1. Das »BKS« ist eine Prüfroutine für Basic-Programme. Bitte mit dem MSE (Seite 159) eingeben.

```
ca88 : ff 38 e9 31 aa fe 4e c2  a5
ca90 : d0 03 fe 50 c2 60 20 c1  df
ca98 : ca a2 00 bd 00 01 f0 03  3d
caa0 : e8 d0 f8 e0 05 b0 14 a0  b2
caa8 : 05 bd 00 01 99 00 01 88  5b
cab0 : ca 10 f6 a9 20 99 00 01  46
cab8 : 88 10 fa 20 0c bf 4c de  b9
cac0 : c4 85 62 86 63 a2 90 38  ae
cac8 : 20 49 bc 4c df bd e0 08  c5
cad0 : d0 06 ad 08 cf d0 01 60  58
cad8 : e0 07 d0 05 ad 0a cf f0  5d
cae0 : f6 e0 11 d0 05 ad 0a cf  2a
cae8 : f0 ed e0 0c 90 09 e0 0f  7b
caf0 : b0 05 ad 0b cf f0 e0 a9  4b
caf8 : 01 9d 0b cf 8e 42 c2 20  cb
cb00 : 22 cb 20 4f c4 ae 4d c2  76
cb08 : e0 08 d0 03 20 12 cb 4c  db
cb10 : 19 ca a9 01 20 54 c4 20  11
cb18 : d2 ff 20 58 c4 a9 14 4c  7f
cb20 : 54 c4 ad 56 c2 d0 01 60  84
cb28 : ad 02 ce cd 3f c2 d0 09  a3
cb30 : ad 03 ce cd 40 c2 d0 01  2b
cb38 : 60 20 4d cb a0 ff c8 f0  84
cb40 : 07 b1 02 99 00 ce d0 f6  7b
cb48 : c0 04 90 f2 60 ae 03 ce  b2
cb50 : e8 d0 01 60 20 4f c4 ad  d8
cb58 : 4d c2 c9 08 d0 26 20 12  5d
cb60 : cb 20 78 cb a0 03 c8 f0  fa
cb68 : 09 b9 00 ce 20 d2 ff aa  16
cb70 : d0 f4 c0 04 90 f0 b0 3a  33
cb78 : ae 02 ce ad 03 ce 20 96  e5
cb80 : ca 4c 3f ab 20 78 cb a0  ec
cb88 : 03 8c 55 c2 8c 57 c2 a9  61
cb90 : 00 ac 57 c2 29 7f 20 d2  c9
cb98 : ff c9 22 d0 08 ad 55 c2  e7
cba0 : 49 ff 8d 55 c2 c8 b9 00  50
cba8 : ce d0 0a c0 04 f0 06 20  99
cbb0 : d7 aa 4c cc ff aa 10 de  dc
cbb8 : c9 ff f0 da 2c 55 c2 30  f1
cbc0 : d5 38 e9 7f aa 8c 57 c2  0e
cbc8 : a0 ff ca f0 08 c8 b9 9e  23
cbd0 : a0 10 fa 30 f5 c8 b9 9e  07
cbd8 : a0 30 b6 20 d2 ff 90 f5  9d
cbe0 : 20 c5 cc ad 21 d0 8d 20  db
cbe8 : d0 20 f6 cb a9 de 8d 09  d9
cbf0 : cf a9 02 4c c3 ff 20 4d  f5
cbf8 : cb 20 4f c4 20 41 cc 20  bf
cc00 : 3e cc a9 76 a0 c1 20 de  34
cc08 : c4 20 3e cc 20 3e cc a9  80
cc10 : 88 a0 c1 20 de c4 ae 4e  c8
cc18 : c2 ad 50 c2 20 cd bd 20  c5
cc20 : 3e cc a9 9c a0 c1 20 de  19
cc28 : c4 ae 4f c2 ad 51 c2 20  20
cc30 : cd bd 20 3e cc 20 58 c4  64
cc38 : 20 54 c4 4c cc ff 20 58  3b
cc40 : c4 ae 4d c2 e0 08 f0 03  1f
cc48 : 4c d7 aa a9 01 20 54 c4  4c
cc50 : a9 02 20 d2 ff 20 58 c4  49
cc58 : a9 14 4c 54 c4 a9 00 8d  5e
cc60 : 43 c2 8d 44 c2 8d 4a c2  37
cc68 : ad 41 c2 8d 5c c2 20 62  39
cc70 : c5 c9 3a f0 22 aa f0 22  46
cc78 : 38 e9 30 c9 0a b0 1e ee  67
cc80 : 4a c2 20 29 c5 b0 0d 6d  49
cc88 : 43 c2 8d 43 c2 90 d9 ee  ee
cc90 : 44 c2 d0 d4 a2 04 60 a2  15
cc98 : 01 60 a2 02 60 a2 03 60  9a
cca0 : a2 04 bd f9 c1 9d 00 cf  9b
cca8 : ca 10 f7 a9 05 8d 05 cf  1e
ccb0 : a9 00 8d 06 cf a9 7b 8d  d1
ccb8 : 09 cf a9 00 a2 1b 9d 0c  a5
ccc0 : cf ca 10 fa 60 a5 02 18  c3
ccc8 : 69 02 90 02 e6 03 c5 2d  8f
ccd0 : d0 07 a5 03 c5 2e d0 01  01
ccd8 : 60 a2 1c 4c ce ca 00 00  5d
```

Listing 1. »BKS« (Schluß)

```
0 AF=28:GOSUB 1000:REM VERSION 7 - GANZ ST
  ARK VERBESSERT                             <040>
1 DIM A$(AF):GOSUB 200:OPEN 4,4:X$="{CTRL-
  H}"+CHR$(13)+"{CTRL-O}                     <002>
2 PRINT#4,"{CTRL-N}------------------------
  --------------{CTRL-O}--                   <152>
3 PRINT#4,"{CTRL-N}BASIC KONTROLL SYSTEM{3
  SPACE}ERKLAERUNGEN{SPACE,CTRL-O}V7         <119>
4 PRINT#4,"{CTRL-N}------------------------
  --------------{CTRL-O}--                   <154>
5 PRINT#4,"OBIGE TABELLE GIBT AUSKUNFT UEB
  ER ALLE LEICHTEN UND SCHWEREN FEHLER,      <012>
6 PRINT#4,"DIE IM GETESTETEN BASIC-PROGRAM
  M AUFGETRETEN SIND.                        <233>
7 PRINT#4:PRINT#4,"BASIC-PROGRAMM{12SPACE}
  : "N$                                      <177>
8 PRINT#4,"TESTDATUM{17SPACE}: "D$           <119>
9 PRINT#4,"VERSION DES TESTPROGRAMMES: "V$   <070>
10 PRINT#4:IF PEEK(52999)THEN 13             <151>
11 PRINT#4,"DER FEHLER NR. 2 (UEBERFLUESSI
   GE LEERZEICHEN IM PROGRAMMTEXT)           <004>
12 PRINT#4,"WURDE NICHT GELISTET, DA ER ZU
    HAEUFIG AUFTRAT.":PRINT#4                <055>
13 RQ$=" RUNTIME-KONSEQUENZEN"+X$            <166>
14 GOSUB 100:PRINT#4,"ERKLAERUNG DER FEHLE
   R:                                        <212>
15 PRINT#4,"TTTTTTTTTTTTTTTTTTTTTT":PRINT#4
   ,"BEISPIEL:                               <035>
16 PRINT#4,"{2SPACE}2720: 17[1] UEBERFLUES
   SIGER BEFEHL                              <022>
17 PRINT#4,"{4SPACE}↑{3SPACE}↑{2SPACE}↑{2S
   PACE}↑"X$;                                <237>
18 PRINT#4,"{4SPACE}_{3SPACE}_{2SPACE}_{2S
   PACE}J ART DES FEHLERS (PAUSCHALTEXT)"X
   $;                                        <183>
19 PRINT#4,"{4SPACE}_{3SPACE}_{2SPACE}_"X$
   ;                                         <215>
20 PRINT#4,"{4SPACE}_{3SPACE}_{2SPACE}J FE
   HLERGRAD: 1 = LEICHTER FEHLER OHNE"RQ$;   <242>
21 PRINT#4,"{4SPACE}_{3SPACE}_{16SPACE}2 =
    SCHWERER FEHLER, WIRD FEHLERMELDUNG"X$
   ;                                         <051>
22 PRINT#4,"{4SPACE}_{3SPACE}_{20SPACE}VER
   URSACHEN"X$;                              <202>
23 PRINT#4,"{4SPACE}_{3SPACE}_"X$;           <192>
24 PRINT#4,"{4SPACE}_{3SPACE}J FEHLERCODEN
   UMMER (SIEHE UNTEN)"X$;                   <005>
25 PRINT#4,"{4SPACE}_"X$;                    <054>
26 PRINT#4,"{4SPACE}J BASIC PROGRAMMZEILEN
   NUMMER                                    <212>
27 PRINT#4                                   <030>
28 PRINT#4,"VERZEICHNIS DER VORKOMMENDEN F
   EHLERCODES MIT BEDEUTUNG:                 <120>
30 PRINT#4,"TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
   TTTTTTTTTTTTTTTTTTTTTTTT                  <093>
40 FOR I=1 TO AF:IF PEEK(53003+I)=.THEN 60   <002>
50 PRINT#4,A$(I)                             <215>
60 NEXT                                      <070>
80 PRINT#4:PRINT#4,"(C) NH-201187-ARR REV7   <029>
99 CLOSE 4:END                               <057>
100 IF PEEK(53000)THEN 110                   <106>
101 PRINT#4,"DER FEHLER NR. 8 (SPRUNG AUF
    REM- ODER TRENNZEILE)                    <099>
102 PRINT#4,"WURDE NICHT GELISTET, DA ER Z
    U HAEUFIG AUFTRAT.":PRINT#4              <145>
110 IF PEEK(53002)THEN 120                   <152>
111 PRINT#4,"DIE FEHLER NR. 7 (GOTO -> STR
    UKTUR) UND NR. 17 (THEN GOTO)            <231>
112 PRINT#4,"WURDEN NICHT GELISTET, DA SIE
     ZU HAEUFIG AUFTRATEN.":PRINT#4          <237>
120 IF PEEK(53003)THEN 130                   <196>
121 PRINT#4,"DIE FEHLER NR. 12, 13 UND 14
    WURDEN NICHT GELISTET.                   <005>
122 PRINT#4                                  <125>
130 RETURN                                   <188>
200 FOR I=1 TO AF:B$=""                      <013>
202 READ A$:IF A$="@"THEN 210                <122>
204 B$=B$+A$+CHR$(13):GOTO 202               <249>
210 A$(I)=LEFT$(B$,LEN(B$)-1):NEXT:RETURN    <190>
1000 FOR I=52992 TO 52996:A$=A$+CHR$(PEEK(
     I)):NEXT:IF A$<>"NSS88"THEN 1100        <160>
1001 IF PEEK(53001)=123 THEN 1102            <230>
1002 IF PEEK(53001)<>222 THEN 1100           <106>
1003 FOR I=.TO AF-1:A=A+PEEK(I+53004):NEXT
     :IF A=.THEN 1104                        <054>
1004 INPUT"{CLR}DATUM{3SPACE}XX.XX.19XX{12
     LEFT}";D$                               <204>
1005 INPUT"NAME PROGRAMM{3SPACE}X{3LEFT}";
     N$                                      <239>
1006 V$=CHR$(48+PEEK(52997))+"."+CHR$(48+P
     EEK(52998))                             <165>
1007 IF V$<>"5.0"THEN 1106                   <166>
1008 RETURN                                  <048>
1100 PRINT"{CLR}BITTE ERSTELLEN SIE ERST E
```

```
     INE LISTE MIT DEM BKS (VERS. >= 5.0)"
     :END                                    <008>
1102 PRINT"{CLR}BITTE UNTERBRECHEN SIE DAS
     BKS PROGRAMM NICHT !":END               <082>
1104 PRINT"{CLR}DAS TESTPROGRAMM ENTHIELT
     KEINE FEHLER !":END                     <175>
1106 PRINT"{CLR}FALSCHE VERSIONSNUMMER !     <100>
2000 DATA 1 = DIREKT NACH DER ZEILENNUMMER
     FOLGT EIN NULLBYTE (DIES WIRD ZU LIS
     T-                                      <098>
2002 DATA"{4SPACE}SCHUTZZWECKEN VERWENDET)
     ",@                                     <146>
2004 DATA 2 = IM PROGRAMMTEXT KOMMT EIN UE
     BERFLUESSIGES LEERZEICHEN VOR,@         <216>
2006 DATA"3 = EIN THEN, GOTO, LIST ETC. BE
     FEHL ZEIGT AUF EINE NICHT EXISTIEREND
     E                                       <138>
2008 DATA"{4SPACE}ZEILE",@                   <005>
2010 DATA 4 = DIE ZEILENNUMMER ALS PARAMET
     ER IST GROESSER ALS 63999,@             <120>
2012 DATA 5 = DIE ZEILENNUMMER ALS PARAMET
     ER ENTHAELT FALSCHE ZEICHEN (BEISPIEL
     S-                                      <124>
2014 DATA"{4SPACE}WEISE GOTO 4+6)",@         <083>
2016 DATA 6 = EINE BASIC ZEILE IST LAENGER
     ALS 255 ZEICHEN,@                       <102>
2018 DATA"7 = EIN GOTO ODER THEN BEFEHL ZE
     IGT AUF EINEN BEFEHL WIE RETURN, GOTO
     ,                                       <144>
2020 DATA"{4SPACE}END USW., DEN MAN AUCH E
     INFACH ANSTELLE DES SPRUNGBEFEHLES HA
     ETTE                                    <029>
2022 DATA"{4SPACE}SETZEN KOENNEN",@          <080>
2024 DATA 8 = EIN SPRUNGBEFEHL ZEIGT AUF E
     INE REM- ODER TRENNZEILE. DIES KANN Z
     U                                       <246>
2026 DATA"{4SPACE}PROBLEMEN BEIM ABTIPPEN
     FUEHREN, WENN DIE ANGESPRUNGENE ZEILE
     WEG-                                    <236>
2028 DATA"{4SPACE}GELASSEN WIRD.",@          <224>
2030 DATA 9 = EINE BASIC ZEILE IST LAENGER
     ALS 255 ZEICHEN,@                       <140>
2032 DATA 10= EIN SPRUNGBEFEHL ZEIGT AUF S
     ICH SELBST (Z.B. 10 GOTO 10),@          <209>
2034 DATA 11= DER BEFEHL 'CONT' DARF NICHT
     IM PROGRAMMTEXT VORKOMMEN,@             <084>
2036 DATA 12= DER BEFEHL 'STOP' SOLLTE NIC
     HT IM PROGRAMMTEXT VORKOMMEN,@          <223>
2038 DATA 13= DER BEFEHL 'NEW' SOLLTE NICH
     T IM PROGRAMMTEXT VORKOMMEN,@           <111>
2040 DATA 14= DER BEFEHL 'LET' SOLLTE NICH
     T IM PROGRAMMTEXT VORKOMMEN,@           <115>
2042 DATA 15= HINTER EINEM REM-BEFEHL STEH
     T EIN GESHIFTETES L (LISTSCHUTZ),@      <228>
2044 DATA 16= EIN ILLEGALES TOKEN KOMMT IM
     PROGRAMMTEXT VOR,@                      <058>
2046 DATA"17= DER BEFEHL 'GOTO' SOLLTE NIC
     HT DIREKT HINTER 'THEN' STEHEN, EINER   <068>
2048 DATA"{4SPACE}VON BEIDEN GENUEGT",@      <220>
2050 DATA 18= HINTER EINEM BEFEHL FEHLT DE
     R PARAMETER,@                           <043>
2052 DATA 19= HINTER GO FEHLT TO,@           <036>
2054 DATA"20= HINTER GOTO, RUN ETC. FOLGEN
     WEITERE BEFEHLE, DIE NIEMALS AUSGE-     <079>
2056 DATA"{4SPACE}FUEHRT WERDEN",@           <219>
2058 DATA 21= EINE ODER MEHRERE KLAMMERN Z
     U VIEL ODER ZU WENIG,@                  <206>
2060 DATA 22= DAS ZEICHEN '↑' ZUR POTENZIE
     RUNG SOLLTE VERMIEDEN WERDEN,@          <000>
2062 DATA 23= DER BEFEHL 'PRINT#' WURDE MI
     T ?# ABGEKUERZT,@                       <079>
2064 DATA 24= FALSCHE REIHENFOLGE DER BASI
     C-ZEILEN,@                              <010>
2066 DATA 25= EIN FALSCHER LINKPOINTER KOM
     MT VOR,@                                <196>
2068 DATA 26= ON OHNE LEGALEN SPRUNGBEFEHL
     ,@                                      <050>
2070 DATA 27= THEN OHNE IF,@                 <037>
2072 DATA 28= DER POINTER 45/46 ZEIGT NICH
     T GENAU AUF DAS ENDE DES BASIC-PRO-     <161>
2074 DATA"{4SPACE}GRAMMES",@                 <252>
2100 :                                       <044>
2102 REM **********************              <127>
2103 REM *                    *              <120>
2104 REM * FRAGEN ?           *              <166>
2106 REM *  NIKOLAUS HEUSLER  *              <182>
2108 REM *  ZWENGAUERWEG 18   *              <118>
2110 REM *  8000 MUENCHEN 71  *              <254>
2112 REM *  TEL. 089/792940   *              <241>
2114 REM *                    *              <131>
2116 REM * REVISED 151287 /NH *              <051>
2117 REM * REVISED 070188 /NH *              <237>
2118 REM *                    *              <135>
2120 REM * (C)OPYRIGHT        *              <254>
2122 REM * NIG SOFTWARE SER-  *              <156>
2123 REM * VICE (NSS),        *              <054>
2124 REM * NH-201187-ARR      *              <138>
2125 REM *                    *              <142>
2126 REM * RELEASED FOR 64'ER *              <084>
2127 REM * LELI-TEST (RS)     *              <153>
2128 REM *                    *              <145>
2130 REM **********************              <155>
```

Listing 2. »BKS.WHAT 5.0« bringt Klarheit in die Fehlermeldungen des BKS. Bitte mit dem Checksummer (Seite 159) eingeben.



Fehler-teufelchen

Haushalt, Sonderheft 36, Seite 14:

Wenn Sie die Fehlermeldung abfangen wollen, die nach Aufruf des Taschenrechners bei der Division durch Null erzeugt wird, korrigieren Sie die Zeile 5170 des Listings:

```
5170 IFE1(I)=0THENRO=2:
ELSEIFRO=2THENRE=E1(I-1)/E1(I)
*FL+E1(0)
```

Arbeiten Sie im 80-Zeichen-Modus, ergänzen Sie bitte Zeile 6830 wie folgt:

```
6830 COLORB,F,H:IFR-
WINDOW(2)=80THENCOLOR(B=0)*
(-6),F:ELSE6840
```

Hi-Eddi, Sonderheft 39, Seite 104:

Ins Listing 1 »HI-EDDI/1351 MAUS« hat sich leider ein Fehler eingeschlichen. Die Zeile 60 lautet korrekt:

```
60 IF A=3 THEN 150
```

MSE, Sonderheft 38 und Sonderheft 39, Seite 160:

Wenn Sie die Zeile 230 so abtippen, wie sie im Sonderheft abgedruckt ist, erhalten Sie als Meldung »Syntax Error«; auch dann, wenn Sie die Zeile völlig korrekt eingegeben haben. Der Grund dafür liegt darin, daß der letzte Buchstabe des Befehls END in die dritte Zeile rutscht. Da der C64 eine Programmzeile über maximal zwei Zeilen akzeptiert, versucht er, das »D« als Befehl zu deuten. Einen solchen Befehl gibt es aber nicht, also beschwert sich der Computer mit der Fehlermeldung.

Die Behebung dieses Fehlers ist sehr einfach: Die erste Möglichkeit besteht darin, alle Leerzeichen zwischen den Befehlen nicht einzugeben. Für die Prüfsumme des Checksummers werden Leerzeichen ohnehin nicht beachtet. Eine zweite Möglichkeit besteht darin, Befehle als Abkürzungen einzugeben. Im Handbuch zum C64 finden Sie eine Übersicht, welche Befehle Sie abgekürzt eingeben können. Beispiel: Statt PRINT geben Sie bei der Eingabe einer Zeile das Fragezeichen »?« ein. Wenn Sie nach Beenden der Eingabe die entsprechende Zeile listen lassen, steht für das Fragezeichen wieder der ausgeschriebene Befehl.

Finden Sie in einem Listing eine Programmzeile, die sich über mehr als zwei Zeilen erstreckt, sollten Sie beim Abtippen immer die Befehle abkürzen und die Leerzeichen zwischen den Befehlen weglassen.

Das Ende der

Wer die öden Fehlermeldungen des Betriebssystems wie »READY« oder »SYNTAX ERROR« satt hat, kann mit dem »Error-Changer« eigene Fehlermeldungen kreieren und auf Diskette speichern.

Selbst geübte Basic-Programmierer machen immer wieder Probeläufe ihrer Programme. Allzuoft stecken noch Fehler im Programm. Das Programm wird korrigiert, neuer Testlauf, vielleicht wieder eine Fehlermeldung, erneute Korrektur ... Und bei jedem Fehler erscheinen die schon tausendfach bekannten, langweiligen Fehlermeldungen des Betriebssystems.

Mit dem Programm »Error-Changer« (Listing 1) wird es auch ohne hervorragende Kenntnisse des Betriebssystems möglich, den Wortlaut vieler Fehlermeldungen frei zu wählen (Bild 1). Das Besondere dabei ist: Ihre neue Fehlermeldung kann bis zu 80 Zeichen umfassen.

Lassen Sie Ihrer Fantasie freien Lauf. Warum sollen alle Meldungen in oft verkürztem Englisch, wie beispielsweise »UNDEFD STATEMENT ERROR«, auf dem Bildschirm erscheinen? Ersetzen Sie diese Meldung beispielsweise in gutem Schullatein durch »QUO VADIS«. Oder bevorzugen Sie deutsche Meldungen wie »SAG SCHON FREUND WOHIN ICH GEHEN SOLL« ?

Eine der häufigsten Meldungen ist das fade »READY«. Wie wäre es mit »STETS ZU DIENSTEN VEREHRTER HERR UND GEBIETER« (beziehungsweise »VEREHRTE HERRIN UND GEBIETERIN«) ? Aus Ihrem sonst so nüchtern reagierenden C64 wird ein ergebener Helfer. Das Einheits-ROM ist out, Individualität ist Trumpf.

Das Programm ist einfach zu bedienen: Laden Sie das Programm mit

```
LOAD "ERROR-CHANGER",8
```

und starten Sie es mit RUN.

Auf dem Bildschirm erscheint nach dem Start der englische Text der ersten Fehlermeldung. Wenn Sie diesen Text übernehmen wollen, drücken Sie die RETURN-Taste. Ansonsten geben Sie den Text ein, den Sie als neue Fehlermeldung bevorzugen. Die Länge des Textes ist nur durch die 80-Zeichen-Begrenzung des INPUT-Befehls be-

JETZT BIN ICH FERTIG PRINT 1/0	READY PRINT 1/0
?NICHT DURCH NULL TEILEN!!! JETZT BIN ICH FERTIG CONT	?DIVISION BY ZERO ERROR READY CONT
?DAS GEHT JETZT NICHT MEHR JETZT BIN ICH FERTIG	?CANT CONTINUE ERROR READY
DIM A(10000)	DIM A(10000)
?ICH HABE KEINEN PLATZ MEHR JETZT BIN ICH FERTIG	?OUT OF MEMORY ERROR READY

Bild 1. Typische Fehlermeldungen im Vergleich: Links die geänderten Meldungen, rechts die Originalmeldungen

```
110 REM ******************************     <122>
130 REM *                            *     <179>
140 REM * E R R O R - C H A N G E R *      <229>
150 REM *                            *     <199>
151 REM *           C 64             *     <066>
152 REM *                            *     <201>
160 REM *     BY ANDREAS KNIPP       *     <105>
170 REM *                            *     <219>
180 REM ******************************     <192>
190 REM                                    <252>
200 POKE 53280,4:POKE 53281,6:GOTO 250     <124>
250 PRINT"{CLR,WHITE}WAEHLBARE FEHLERMELDU
    NGEN BY A.KNIPP KPS":DIM T$(40):FZ=1   <197>
260 T$(30)="BREAK ERROR"                   <104>
270 T$(31)="READY.":HI=256                 <150>
280 FOR I=49152 TO 49232                   <139>
300 READ X:POKE I,X:NEXT                   <225>
310 POKE 768,139:POKE 769,227:REM FEHLERME
    LDUNGEN AUF ROM  STELLEN.              <202>
320 LV=41372:SV=49293:VA=49233             <210>
330 FOR LV=41373 TO 41767:REM EINLESEN DER
     ALTEN FEHLERMELDUNGEN                 <043>
340 T$(FZ)=T$(FZ)+CHR$(PEEK(LV)AND 127):IF
     PEEK(LV)<127 THEN 370                 <236>
350 T$(FZ)=T$(FZ)+" ERROR":PRINT T$(FZ)    <052>
360 FZ=FZ+1                                <100>
370 NEXT LV:FZ=FZ+1:REM PLATZ FUER READYME
    LDUNG                                  <061>
380 FOR I=1 TO FZ:REM MOEGLICHKEIT DER AEN
    DERUNG                                 <126>
390 PRINT"{2RIGHT}"T$(I)CHR$(13)"{UP}";:IN
    PUT T$(I)                              <152>
400 IF LEN(T$(I))<2 THEN 390               <146>
410 NEXT I                                 <240>
420 INPUT"ALLES RICHTIG J/N";AR$:IF AR$<>"
    J"THEN 380                             <054>
430 FOR J=1 TO FZ-1:REM NEUE FM IN DEN SPE
    ICHER SCHREIBEN                        <109>
440 FOR I=1 TO LEN(T$(J))-1                <031>
450 POKE SV+I-1,ASC(MID$(T$(J),I,1))       <061>
460 NEXT I                                 <034>
470 POKE SV+I-1,ASC(MID$(T$(J),I,1))+128   <126>
480 A=SV:GOSUB 690:POKE VA+1,AH%:POKE VA,A
    L%                                     <187>
490 VA=VA+2:SV=SV+I:NEXT J                 <038>
500 POKE SV,13                             <189>
510 FOR J=1 TO LEN(T$(31)):POKE SV+J,ASC(M
    ID$(T$(31),J,1)):NEXT:REM READY        <167>
520 POKE SV+J,13:A=SV:GOSUB 690:POKE 49220
    ,AL%:POKE 49222,AH%:POKE SV+J+1,0      <085>
530 SYS 49152                              <080>
540 INPUT"SPEICHERUNG J/N";S$:IF S$="N"THE
    N END                                  <008>
550 IF S$<>"J"THEN 540                     <069>
560 SA=49152:AE=SV+J+2                     <207>
570 PRINT"GERAETENUMMER";                  <238>
580 PRINT"{3SPACE}FLOPPY = Z.B.{2SPACE}8{2
    4SPACE}DATASETTE ={4SPACE}1";          <196>
590 INPUT DN                               <110>
600 A$="@:FEHLER.OBJ":A%=LEN(A$)           <056>
610 A=681:GOSUB 690:POKE 183,A%:POKE 187,A
    L%:POKE 188,AH%                        <122>
620 FOR I=1 TO A%:POKE 680+I,ASC(MID$(A$,I
    )):NEXT:REM FILENAME                   <204>
630 A=SA:GOSUB 690:POKE 251,AL%:POKE 252,A
    H%:REM STARTADRESSE                    <252>
640 A=AE:GOSUB 690:POKE 781,AL%:POKE 782,A
    H%:REM ENDADRESSE                      <109>
650 POKE 186,DN:POKE 780,251:SYS 65496:REM
     SAVE:DN = DEVICENUMMER                <027>
660 PRINT"{WHITE}SPEICHERUNG ERFOLGTE UNTE
    R "CHR$(34)"FEHLER.OBJ"CHR$(34)        <133>
670 PRINT"ABSOLUT LADEN UND MIT SYS 49152
    STARTEN"                               <254>
680 END                                    <174>
690 AH%=A/HI:AL%=A-AH%*HI                  <231>
700 RETURN                                 <250>
710 DATA 169,11,160,192,141,0,3,140,1,3,96
    ,138                                   <111>
720 DATA 16,3,76,67,192,10,170,189,79,192,
    133,34                                 <186>
730 DATA 189,80,192,133,35,32,204,255,169,
    0,133,19                               <008>
740 DATA 32,215,170,32,69,171,160,0,177,34
    ,72,41                                 <019>
750 DATA 127,32,71,171,200,104,16,244,32,1
    22,166,164                             <132>
760 DATA 58,200,240,3,32,194,189,169,187,1
    60,194,76                              <174>
770 DATA 120,164                           <009>
780 DATA 169,139,160,227,76,4,192          <192>
```

Listing 1. »Error-Changer« bitte mit dem Checksummer eingeben

langweiligen Fehlermeldungen

schränkt. Sollte der neue Text kürzer sein als der alte, überschreiben Sie den Rest mit Leerzeichen.

Sie können auch die Abbruchmeldung BREAK oder die READY-Meldung ändern. Bedingt durch das Betriebssystem kommt es jedoch gelegentlich vor, daß nicht immer der geänderte Text erscheint.

Bei der Eingabe sollten Sie darauf achten, Sonderzeichen wie Komma oder Doppelpunkt zu vermeiden. Der INPUT-Befehl ignoriert diese Sonderzeichen.

Haben Sie alle Fehlermeldungen, die vom Programm vorgegeben werden, geändert beziehungsweise übernommen, werden nach einer Sicherheitsabfrage die neuen Meldungen in den Speicher des C 64 zurückgeschrieben. Die Änderungen sind natürlich nur so lange wirksam, bis der Computer ausgeschaltet oder ein Reset ausgeführt wird.

Das Programm fragt Sie anschließend, ob die neuen Fehlermeldungen auf Diskette oder Datasette gespeichert werden sollen. Der Dateiname »FEHLER.OBJ« ist vorgegeben. Existiert bereits eine Datei mit diesem Namen, wird sie automatisch überschrieben.

Wollen Sie Ihre kreativen Einfälle wieder in den Computer laden, geben Sie bitte ein:

```
LOAD "FEHLER.OBJ",8,1
```

Datasettenbenutzer ersetzen bitte die 8 durch eine 1.

Nach dem Laden ist unbedingt der Befehl NEW einzugeben. Die neuen Fehlermeldungen werden mit SYS 49152 aktiviert und mit SYS 49226 ausgeschaltet.

Hinweise für Programmierer:

In den Speicherstellen 768 und 768 steht ein »Zeiger« auf die Stellen des Betriebssystems, wo die Fehlermeldungen des C 64 zu finden sind. Dieser Zeiger, der die Adresse für den Beginn der Meldungen darstellt, wird auf einen neuen Bereich eingestellt, in dem Ihre neuen Fehlermeldungen gespeichert werden. Tritt ein Fehler ein, wird anhand der Fehlernummer und der geänderten Adresse zu Ihrem neuen Text gesprungen und dieser ausgegeben. (Andreas Knipp/ef)

Kurzinfo: Error-Changer

Programmfunktion: Eigene Texte für Fehlermeldungen des C 64
Laden: LOAD "ERROR-CHANGER",8
Starten: Nach dem Laden RUN eingeben.
Besonderheiten: Neue Fehlermeldungen können auf Diskette oder Datasette gespeichert werden. Der Name »FEHLER.OBJ« wird vom Programm vorgegeben. Zum Laden geben Sie LOAD "FEHLER.OBJ",8,1 (für Datasette die 8 durch 1 ersetzen) ein. Nach dem Laden ist unbedingt NEW einzugeben. Aktiviert werden die Meldungen mit SYS 49152, ausgeschaltet mit SYS 49226.
Programmautor: Andreas Knipp

Diät für Basic-Programme

Der »Basic-Packer« macht auch Ihre Programme klein. Sie sparen viel Speicher-Platz, weil alle überflüssigen Bytes (z.B. Leerzeichen) aus Ihrer Software entfernt werden.

(Fortsetzung auf Seite 40)

Illustration: Rolf Boyke

Dieses Programm verkürzt jedes Basic-Programm, indem es nicht benötigte Leerzeichen, REM-Zeilen und so weiter aus dem Programm entfernt. Zusätzlich können als Option die Basic-Zeilen, die Editor-bedingt nur 80 Zeichen lang sein können, auf bis zu 255 Zeichen pro Zeile zusammengeschoben werden. Dies ist gelegentlich notwendig, wenn Sie zum Beispiel ein Adreß- oder Dateiverwaltungsprogramm geschrieben haben und es zur besseren Übersichtlichkeit mit REM-Zeilen gut dokumentierten. Im nachhinein stellen Sie aber fest, daß Ihnen Ihr Speicher für die Variablen nicht mehr ganz ausreicht (OUT OF MEMORY ERROR) und Sie dringend Platz benötigen. In diesem Falle hilft es Ihnen, das Hauptprogramm zu verkürzen, indem Sie alle nicht benötigten Bytes entfernen. Dies sind REM-Zeilen, nicht benötigte Spaces (Leerzeichen) oder ähnliches. Ebenfalls hilft es, die Basic-Zeilen neu zu »linken«, also mehrere Zeilen zu einer einzigen zusammenzufügen. Hierbei entfallen pro gesparter Zeilennummer 4 Byte für die Zeilennummer und den Linkpointer auf die nächste Zeile. Dadurch kann oft sehr viel Speicherplatz gespart werden.

Kurzinfo: Basic-Packer

Programmart: Hilfsprogramm für Basic-Programmierer
Laden: LOAD "BASIC-PACKER",8
Starten: Nach dem Laden RUN eingeben.
Besonderheiten: Nach dem Start wird das Programm nach $C000 verschoben. Laden Sie jetzt das Basic-Programm, das Sie kürzen wollen. Mit SYS 49152 wird der »Basic-Packer« aktiviert.
Programmautor: Christoph Zwerschke

Nach dem Packen ist das Programm zwar kaum noch editierbar, und die Lesbarkeit des Programms hat auch gelitten, aber es läuft genauso wie das Original (meist sogar etwas schneller, da der Interpreter jetzt nicht mehr soviel Programm »durchforsten« muß). Außerdem hat man noch Speicherplatz gespart, und das ist bei vielen Basic-Programmen ja auch der Effekt, den man erreichen will. Bevor Sie allerdings ein Basic-Programm mit dem »Basic-Packer« verkürzen, sollten Sie es sich noch einmal genau anschauen. Wenn Sie beispielsweise eine REM-Zeile als Sprungziel für ein GOTO oder GOSUB gewählt haben, kann es zu Schwierigkeiten im Programmablauf kommen.

Eine Hilfe zum Herausfinden solcher problematischen Programmteile bietet das »Basic-Kontroll-System«, das Sie in diesem Heft ab Seite 24 finden. Das BKS ermittelt sehr schnell, ob Sie solche Sprungziele verwenden.

Eingabehinweise

Das Programm (Listing 1) geben Sie bitte mit Hilfe des MSE ein und speichern es auf Ihren Datenträger.

Danach läßt sich der Basic-Packer normal mit »LOAD"BASIC-PACKER",8« laden und durch RUN starten. Er verschiebt sich selbständig in den $C000-Bereich. Sie können jetzt das Basic-Programm, das verkürzt werden soll, von Diskette laden. Mit SYS 49152 wird der Packer aktiviert. Anschließend haben Sie die Wahl, ob Sie nur alle Leerzeichen und REM-Zeilen löschen oder zusätzlich alle Zeilen auf eine maximale Länge bringen wollen. Im zweiten Fall ist ein Editieren der Zeilen nicht mehr möglich.

Nach Abschluß des Packens wird Ihnen noch angezeigt, wie viele Bytes eingespart wurden. Jetzt ist das Programm gepackt und bereit zum Speichern.

(Christoph Zwerschke/ef)


```
Name : basic-packer        0801 0f48
-----------------------------------
0801 : 23 08 00 00 9e 32 30 38  d5
0809 : 38 14 14 14 14 14 14 14  2d
0811 : 14 14 42 41 53 49 43 20  b5
0819 : 43 4f 4e 44 45 4e 53 45  bf
0821 : 52 00 00 00 00 00 00 a2  b9
0829 : 56 a0 08 86 5f 84 60 a2  83
0831 : 13 a0 0f 86 5a 84 5b a2  a5
0839 : bd a0 c6 86 58 84 59 20  18
0841 : bf a3 a0 00 b9 13 0f 20  ab
0849 : d2 ff c8 c0 33 d0 f5 20  37
0851 : 44 a6 6c 02 a0 a9 0b 8d  e2
0859 : 20 d0 8d 21 d0 a2 ff a0  cc
0861 : c4 20 99 c0 20 d3 c4 a0  a9
0869 : 01 b1 2b d0 0a a2 6a a0  c8
0871 : c6 20 99 c0 6c 02 a0 a5  6a
0879 : 2d 8d 44 03 a5 2e 8d 45  6b
0881 : 03 a2 58 a0 c5 20 99 c0  45
0889 : a9 00 85 c6 20 e4 ff c9  29
0891 : 03 f0 3b c9 41 f0 04 c9  54
0899 : 42 d0 f1 8d 46 03 20 d2  14
08a1 : ff a2 8d a0 c6 20 99 c0  be
08a9 : a9 2a 8d 43 03 78 20 af  07
08b1 : c0 20 d3 c4 20 a8 c1 20  9d
08b9 : d3 c4 ad 46 03 c9 41 f0  88
08c1 : 0c 20 5b c3 20 d3 c4 20  21
08c9 : a8 c1 20 d3 c4 58 a2 a4  b7
08d1 : a0 c6 20 99 c0 ad 44 03  a0
08d9 : 38 e5 2d aa ad 45 03 e5  81
08e1 : 2e 20 cd bd a2 aa a0 c6  da
08e9 : 20 99 c0 6c 02 a0 86 7a  c8
08f1 : 84 7b a0 00 b1 7a f0 0b  24
08f9 : 20 d2 ff e6 7a d0 f5 e6  33
0901 : 7b d0 f1 60 a5 2b 85 7a  2b
0909 : a5 2c 85 7b 20 ef c4 a5  75
0911 : 7a 38 e9 01 85 41 a5 7b  32
0919 : e9 00 85 42 a0 04 20 71  3a
0921 : c4 a0 00 20 90 c1 b1 7a  0c
0929 : d0 03 4c 6f c1 c9 3a f0  b1
0931 : 46 d0 05 20 90 c1 b1 7a  f8
0939 : c9 00 d0 0e c8 c8 b1 7a  87
0941 : d0 01 60 88 20 71 c4 4c  f4
0949 : b7 c0 c9 3a d0 30 20 71  0c
0951 : c4 a5 7a 85 41 a5 7b 85  71
0959 : 42 a0 01 20 90 c1 b1 7a  03
0961 : f0 04 c9 3a d0 18 a5 41  f4
0969 : 85 7a a5 42 85 7b a0 01  96
0971 : 20 7c c4 a0 00 f0 bf c8  2d
0979 : 20 7c c4 4c cc c0 c9 22  d0
0981 : d0 1a c8 b1 7a f0 06 c9  a1
0989 : 22 d0 f7 f0 54 20 71 c4  c5
0991 : 20 a6 c4 a0 00 a9 22 91  42
0999 : 7a c8 d0 a0 c9 83 d0 20  fc
09a1 : c8 20 90 c1 b1 7a f0 94  b1
09a9 : c9 3a f0 a2 c9 2c f0 f0  c3
09b1 : c9 22 d0 09 c8 b1 7a f0  c6
09b9 : d4 c9 22 d0 f7 c8 d0 e4  e7
09c1 : c9 8f d0 1d a5 41 85 7a  99
09c9 : a5 42 85 7b a0 00 b1 7a  26
09d1 : d0 02 a0 04 c8 b1 7a d0  f1
09d9 : fb 20 7c c4 a0 00 4c e7  a7
09e1 : c0 c8 4c de c0 b1 7a c9  0b
09e9 : 20 d0 11 20 71 c4 a0 00  79
09f1 : a9 20 c8 d1 7a f0 fb 20  76
09f9 : 7c c4 a0 00 60 a9 00 8d  6e
0a01 : 40 03 a9 01 8d 3f 03 8d  47
0a09 : 41 03 20 8e a6 20 ef c4  5a
0a11 : a0 02 b1 7a d0 03 4c a2  0a
0a19 : c2 a5 7a 18 69 04 85 7a  11
0a21 : 90 02 e6 7b 20 73 00 a0  ba
0a29 : 00 b1 7a f0 e0 c9 22 d0  45
0a31 : 0e 20 73 00 c9 22 f0 ec  77
0a39 : c9 00 f0 d1 4c dc c1 c9  bf
0a41 : 89 f0 17 c9 8a f0 13 c9  51
0a49 : 8d f0 0f c9 a7 f0 0b c9  0d
0a51 : cb d0 d1 20 73 00 c9 a4  a4
0a59 : d0 cd 20 73 00 90 06 c9  b7
0a61 : 2c f0 f7 d0 c2 a5 7a a6  ae
0a69 : 7b 85 31 86 32 20 79 00  ce
0a71 : 20 6b a9 20 be c2 a0 01  3c
0a79 : b1 2f f0 ab c8 b1 2f 38  bb
0a81 : e5 14 c8 b1 2f e5 15 b0  b0
0a89 : 06 20 d3 c2 4c 21 c2 a5  11
0a91 : 7a e5 31 8d 3c 03 ad 3d  09
0a99 : 03 ae 3e 03 85 63 86 62  36
0aa1 : a2 90 38 20 49 bc 20 dd  54
0aa9 : bd a0 ff c8 b9 01 01 d0  18
0ab1 : fa cc 3c 03 f0 1f 90 0d  e5
0ab9 : 98 ed 3c 03 8d 42 03 20  ef
0ac1 : ef c2 4c 80 c2 ad 3c 03  c5
0ac9 : 8c 3c 03 38 ed 3c 03 8d  23
0ad1 : 42 03 20 1e c3 a0 00 b9  15
0ad9 : 01 01 f0 06 91 31 c8 4c  b6
0ae1 : 82 c2 ac 42 03 f0 03 20  3c
0ae9 : 33 a5 20 79 00 c9 2c f0  07
0af1 : 03 4c d2 c1 4c 05 c2 20  3f
0af9 : be c2 a0 01 b1 2f f0 12  dd
0b01 : c8 ad 3d 03 91 2f c8 ad  61
0b09 : 3e 03 91 2f 20 d3 c2 4c  57
0b11 : a5 c2 60 ad 3f 03 ae 40  2c
0b19 : 03 8d 3d 03 8e 3e 03 a5  c5
0b21 : 2b a6 2c 85 2f 86 30 60  04
0b29 : a0 01 b1 2f aa 88 b1 2f  b0
0b31 : 85 2f 86 30 ad 3d 03 18  f6
```

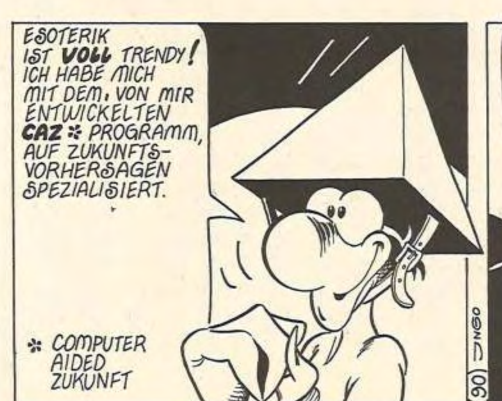
```
0b39 : 6d 41 03 8d 3d 03 90 03   ee
0b41 : ee 3e 03 60 a5 2d a6 2e   d6
0b49 : 85 5a 86 5b 18 6d 42 03   04
0b51 : 85 58 85 2d 90 01 e8 86   cb
0b59 : 59 86 2e a5 7a a6 7b 85   0b
0b61 : 5f 86 60 20 bf a3 a5 7a   c4
0b69 : 18 6d 42 03 85 7a 90 02   9b
0b71 : e6 7b 60 a5 2d a6 2e 38   13
0b79 : ed 42 03 b0 01 ca 85 2d   35
0b81 : 86 2e 38 a5 7a a6 7b ed   88
0b89 : 42 03 b0 01 ca 85 7a 86   69
0b91 : 7b 85 22 86 23 c5 2d 90   5e
0b99 : 04 e4 2e b0 d5 ac 42 03   83
0ba1 : b1 22 a0 00 91 22 a4 22   8c
0ba9 : c8 d0 01 e8 98 4c 3c c3   9b
0bb1 : a5 2b 85 7a a5 2c 85 7b   65
0bb9 : a9 00 85 3e 20 ef c4 a0   61
0bc1 : 03 c8 b1 7a f0 0f c9 22   d7
0bc9 : d0 24 c8 b1 7a f0 06 c9   ee
0bd1 : 22 d0 f7 f0 ec a5 3e f0   4e
0bd9 : 08 a9 fa 91 7a a9 00 85   a7
0be1 : 3e c8 c8 b1 7a f0 63 88   b9
0be9 : 20 71 c4 4c 67 c3 c9 fa   2e
0bf1 : f0 e3 c9 89 f0 18 c9 8d   88
0bf9 : f0 14 c9 9b f0 10 c9 8a   a5
0c01 : f0 0c c9 a7 f0 08 c9 8b   ec
0c09 : d0 b7 85 3e f0 b3 c8 20   ee
0c11 : 71 c4 a0 00 20 79 00 c9   6e
0c19 : 3a b0 a7 c9 30 90 a3 20   25
0c21 : 79 00 20 6b a9 20 13 a6   45
0c29 : b0 10 a0 00 b1 7a c9 2c   78
0c31 : d0 90 20 73 00 a0 ff 4c   5d
0c39 : b9 c3 a5 5f d0 02 c6 60   22
0c41 : c6 5f a0 00 a9 fa 91 5f   56
0c49 : d0 e0 a5 2b 85 7a a5 2c   73
0c51 : 85 7b 20 ef c4 a0 03 c8   89
0c59 : b1 7a f0 04 c9 fa d0 f7   ac
0c61 : c8 98 a0 00 91 7a c8 91   d1
0c69 : 7a a8 20 71 c4 a0 01 b1   26
0c71 : 7a d0 df a5 2b 85 7a a5   14
0c79 : 2c 85 7b a0 04 b1 7a f0   f4
0c81 : 07 c9 fa f0 36 c8 d0 f5   22
0c89 : 20 ef c4 84 3e c8 c8 b1   13
0c91 : 7a d0 01 60 18 65 3e 90   87
0c99 : 05 a4 3e 4c 66 c4 a5 7b   24
0ca1 : 85 40 a5 7a 18 65 3e 85   b0
0ca9 : 3f 90 02 e6 40 a0 04 20   e7
```

```
0cb1 : 84 c4 a4 3e a9 3a 91 7a   30
0cb9 : 4c 28 c4 a9 00 91 7a c8   88
0cc1 : 20 71 c4 4c 26 c4 98 18   70
0cc9 : 65 7a 85 7a 90 02 e6 7b   c8
0cd1 : 60 a5 7a 85 3f a5 7b 85   6d
0cd9 : 40 84 3d b1 3f a0 00 91   fd
0ce1 : 3f a4 3d e6 3f d0 f4 e6   bb
0ce9 : 40 a5 2e c5 40 b0 ec a5   c9
0cf1 : 2d 38 e5 3d 85 2d b0 02   e4
0cf9 : c6 2e 60 a5 2d 85 3f a5   ea
0d01 : 2e 85 40 e6 2d d0 02 e6   0e
0d09 : 2e a0 00 b1 3f c8 91 3f   bd
0d11 : a5 3f c5 7a d0 06 a5 40   6b
0d19 : c5 7b f0 0b a5 3f d0 02   d5
0d21 : c6 40 c6 3f 4c b4 c4 60   df
0d29 : 20 8e a6 a9 00 a8 91 7a   f0
0d31 : 20 33 a5 a5 23 85 2e 98   51
0d39 : 38 65 22 85 2d 90 02 e6   8a
0d41 : 2e 4c 60 a6 a9 9d 20 d2   30
0d49 : ff ad 43 03 49 0a 8d 43   f2
0d51 : 03 4c d2 ff 93 0e 08 9e   36
0d59 : 11 11 1d c2 c1 d3 c9 c3   fc
0d61 : 20 c3 cf ce c4 c5 ce d3   8e
0d69 : c5 d2 2c 20 31 39 38 34   cd
0d71 : 20 42 59 20 c3 48 2e 20   84
0d79 : da 57 45 52 53 43 48 4b   a1
0d81 : 45 1d 60 60 60 60 60 60   c4
0d89 : 60 60 60 60 60 60 60 60   89
0d91 : 60 60 60 60 60 60 60 60   91
0d99 : 60 60 60 60 60 60 60 60   99
0da1 : 60 60 60 60 60 60 60 60   a1
0da9 : 60 60 0d 11 00 41 29 20   8e
0db1 : cc 4f 45 53 43 48 54 20   e9
0db9 : 55 4e 57 49 43 48 54 49   8f
0dc1 : 47 45 20 cc 45 45 52 5a   c9
0dc9 : 45 49 43 48 45 4e 20 55   7f
0dd1 : 4e 44 0d 1d 1d 1d da 45   d9
0dd9 : 49 4c 45 4e 20 28 41 55   56
0de1 : 43 48 20 41 4c 4c 45 20   f5
0de9 : 22 d2 c5 cd 22 2d da 45   21
0df1 : 49 4c 45 4e 29 2e 0d 1d   ee
0df9 : 1d 1d ce 45 55 4e 55 4d   b9
0e01 : 45 52 49 45 52 55 4e 47   02
0e09 : 20 41 4c 4c 45 52 20 da   84
0e11 : 45 49 4c 45 4e 2e 0d 11   63
0e19 : 42 29 20 d7 49 45 20 41   b5
0e21 : 29 2c 20 41 42 45 52 20   68
```

```
0e29 : 44 49 45 20 da 45 49 4c   fd
0e31 : 45 4e 20 57 45 52 44 45   13
0e39 : 4e 20 5a 55 52 0d 1d 1d   15
0e41 : 1d 4d 41 58 49 4d 41 4c   fd
0e49 : 45 4e 20 cc 41 45 4e 47   5d
0e51 : 45 20 5a 55 53 41 4d 4d   f7
0e59 : 45 4e 47 45 46 41 53 53   a2
0e61 : 54 2e 0d 1d 1d 1d d6 4f   68
0e69 : 52 53 49 43 48 54 3a 20   70
0e71 : c4 41 53 20 d0 52 4f 47   1a
0e79 : 52 41 4d 4d 20 4b 41 4e   67
0e81 : 4e 20 44 41 4e 4e 0d 1d   de
0e89 : 1d 1d 4e 49 43 48 54 20   fa
0e91 : 4d 45 48 52 20 45 44 49   ad
0e99 : 54 49 45 52 54 20 57 45   5c
0ea1 : 52 44 45 4e 2e 0d 11 c2   46
0ea9 : 49 54 54 45 20 45 49 4e   c8
0eb1 : 47 45 42 45 4e 20 28 41   dd
0eb9 : 2f 42 29 20 3f 1d 00 cb   cc
0ec1 : 45 49 4e 20 c2 c1 d3 c9   60
0ec9 : c3 2d d0 52 4f 47 52 41   9c
0ed1 : 4d 4d 20 49 4d 20 d3 50   bc
0ed9 : 45 49 43 48 45 52 20 21   46
0ee1 : 0d 00 0d 11 c2 45 49 20   10
0ee9 : 44 45 52 20 c1 52 42 45   ab
0ef1 : 49 54 20 2e 2e 2e 20 2a   5b
0ef9 : 00 0d 11 d5 4d 20 00 20   95
0f01 : c2 59 54 45 53 20 47 45   0b
0f09 : 4b 55 45 52 5a 54 2e 0d   b6
0f11 : 09 00 0d 2a 2a 2a 20 42   9c
0f19 : 41 53 49 43 20 43 4f 4e   b5
0f21 : 44 45 4e 53 45 52 20 2a   c2
0f29 : 2a 2a 0d 0d 53 54 41 52   cf
0f31 : 54 45 4e 20 4d 49 54 20   70
0f39 : 22 53 59 53 20 34 39 31   b1
0f41 : 35 32 22 2e 0d 00 00 02   b3
```

Listing 1. »Basic-Packer« – ein Programm, um Basic-Listings kürzer und schneller zu machen. Bitte mit dem MSE (Seite 159) eingeben.









Im Sonderheft 39 auf Seite 27 haben wir es angekündigt: Hier ist das Konvertierprogramm, mit dem Sie problemlos Texte vom Mastertext- ins Giga-Publish-Format umwandeln können. Außerdem finden Sie noch einige interessante Hinweise zur effektiven Arbeit mit Giga-Publish.

Giga-P Texte kon

Mastertext verwendet ein spezielles Format beim Speichern, das in Giga-Publish nicht eingelesen werden kann. »Master/Konvert« beseitigt dieses Problem. Es wandelt den Text in ein Format um, das der Text-Editor von Giga-Publish laden und verarbeiten kann.

Das Hilfsprogramm »MASTER/KONVERT« (Listing 1), in Verbindung mit der Routine »tc0« (Listing 2), wird benötigt, wenn Sie eine Datei unter Mastertext erstellt haben.

Hinweise zur Bedienung:

Wenn Sie die beiden Listings mit dem MSE abgetippt haben, speichern Sie die beiden Programme auf eine Diskette. Kopieren Sie dann mit einem geeigneten File-Kopierprogramm die beiden Giga-Publish-Dateien »gpF« und »gpM« auf dieselbe Diskette. Ein Tip: Wenn Sie kein Kopierprogramm haben, verwenden Sie einfach den MSE zum Kopieren. Laden Sie eine Datei in den MSE, legen anschließend die neue Diskette ins Laufwerk und speichern die Datei zum Schluß.

Auf der Diskette sollten nun folgende vier Dateien sein:
- master/konvert
- tc0
- gpF
- gpM

Jetzt können Sie das Programm mit »LOAD "MASTER/KONVERT", 8,1« laden. Es startet automatisch.

Das Programm meldet sich mit einem Auswahlmenü, das folgende Optionen hat:

a) laden
b) speichern
x) exit

Legen Sie bitte die Diskette mit Ihrer Mastertext-Datei ins Laufwerk, bevor Sie einen Menüpunkt auswählen.


```
Name : master/konvert     0324 03ca
-----------------------------------
0324 : 3c 03 ca f1 ed f6 3e f1   46
032c : 2f f3 66 fe a5 f4 ed f5   74
0334 : d8 a2 ff 9a 78 20 53 e4   50
033c : 20 44 e5 a9 57 a2 f1 8d   9a
0344 : 24 03 8e 25 03 a2 00 bd   f3
034c : 57 03 9d 00 c0 ca d0 f7   22
0354 : 4c 00 c0 a9 00 85 9d 8d   c3
035c : 20 d0 8d 21 d0 a9 08 aa   3c
0364 : a8 20 ba ff a9 03 a2 67   d7
036c : a0 c0 20 bd ff a9 00 20   ba
0374 : d5 ff 18 ad 14 c0 69 03   f8
037c : 8d 14 c0 c9 73 d0 e5 a9   25
0384 : 03 a2 64 a0 c0 20 bd ff   09
038c : 20 c0 ff a2 08 20 c6 ff   fd
0394 : 20 a5 ff 20 a5 ff a2 00   70
039c : 20 a5 ff 9d fc 1e e8 d0   48
03a4 : f7 20 a5 ff 9d fc 1f e8   25
03ac : e0 04 d0 f5 20 cc ff a9   3d
03b4 : 08 20 c3 ff 4c 00 20 47   91
03bc : 50 cd 47 50 c6 47 50 ce   54
03c4 : 54 43 30 54 43 31 00 00   0e
```

Listing 1. »Master/Konvert« bitte mit dem MSE eingeben

```
Name : tc0                2000 2c0a
-----------------------------------
2000 : 4c 32 2b 00 00 00 00 00   30
2008 : 00 00 ff 01 02 03 04 05   7b
2010 : 06 07 08 09 0a 0b 0c 0d   00
2018 : 0e 0f 10 11 12 13 14 15   08
2020 : 16 17 18 19 1a 5b 5c 5d   94
2028 : 5e ff 20 21 22 00 24 25   af
2030 : 26 27 28 29 2a 2b 2c 2d   20
2038 : 2e 2f 30 31 32 33 34 35   28
2040 : 36 37 38 39 3a 3b 3c 3d   30
2048 : 3e 3f 40 41 42 43 44 45   38
2050 : 46 47 48 49 4a 4b 4c 4d   40
2058 : 4e 4f 50 51 52 53 54 55   48
2060 : 56 57 58 59 5a 1b 1c 1d   cc
2068 : ff ff ff ff ff ff ff ff   67
2070 : ff ff ff ff ff ff ff ff   6f
2078 : ff ff ff ff ff ff ff ff   77
2080 : ff ff ff ff ff ff ff ff   7f
2088 : 23 81 ff ff ff ff ff ff   6c
2090 : ff ff ff ff ff ff ff ff   8f
2098 : 80 ff ff ff ff ff ff ff   18
20a0 : ff ff ff ff ff ff ff ff   9f
20a8 : ff ff ff ff ff ff ff ff   a7
20b0 : ff ff ff ff ff ff ff ff   af
20b8 : ff ff 86 04 85 05 a9 00   00
20c0 : a2 04 9d 00 02 ca 10 fa   78
20c8 : a2 10 86 02 a2 05 a0 00   29
20d0 : 18 b9 00 02 2a c9 0a 90   3f
20d8 : 02 e9 0a 99 00 02 c8 ca   4d
20e0 : d0 ef 06 04 26 05 90 15   a1
20e8 : bd 00 02 18 69 01 c9 0a   03
20f0 : 90 08 a9 00 9d 00 02 e8   a2
20f8 : d0 ee 9d 00 02 c6 02 d0   a7
2100 : cb ad 00 02 ae 04 02 8d   10
2108 : 04 02 8e 00 02 ad 01 02   47
2110 : ae 03 02 8d 03 02 8e 01   ef
2118 : 02 a2 04 bd 00 02 09 30   b9
2120 : 9d 00 02 ca 10 f5 a2 00   d2
2128 : bd 00 02 c9 30 d0 0a a9   a4
2130 : 7f 9d 00 02 e8 e0 04 d0   05
2138 : ef 60 a6 d6 bd 75 21 85   f3
2140 : d1 bd 8e 21 85 d2 60 a2   6d
2148 : 18 20 3c 21 a0 27 a9 7f   8c
2150 : 91 d1 88 10 fb ca 10 f1   28
2158 : e8 a9 0f 9d 00 d8 9d 00   ca
2160 : d9 9d 00 da 9d e7 da ca   7d
2168 : d0 f1 a9 0b 8d 20 d0 a9   6d
2170 : 00 8d 21 d0 60 00 28 50   e1
2178 : 78 a0 c8 f0 18 40 68 90   d7
2180 : b8 e0 08 30 58 80 a8 d0   7e
2188 : f8 20 48 70 98 c0 04 04   58
2190 : 04 04 04 04 04 05 05 05   9f
2198 : 05 05 05 06 06 06 06 06   d7
21a0 : 06 06 07 07 07 07 07 84   1a
21a8 : d6 86 d3 8e cc 22 8d ce   3a
21b0 : 22 29 3f 18 65 d3 8d cd   00
21b8 : 22 20 3a 21 2c ce 22 50   ff
21c0 : 0d ac cd 22 88 a9 7f 91   d2
21c8 : d1 cc cc 22 d0 f6 20 83   c3
21d0 : 24 20 e5 24 20 38 24 b0   b8
21d8 : fb a2 07 dd fe 21 f0 0a   73
21e0 : ca 10 f8 c9 70 b0 ed 4c   06
21e8 : 80 22 a8 8a 0a aa bd 06   ee
21f0 : 22 8d fc 21 bd 07 22 8d   f4
21f8 : fd 21 98 4c ff ff f1 80   fe
```

ublish:

vertieren

Mit <a> **(laden)** wird der Text geladen: Das Directory der eingelegten Diskette wird eingelesen und auf dem Bildschirm angezeigt. Mit den Cursortasten kann es auf- und abgescrollt werden. Steht Ihr Text genau zwischen den Begrenzungszeichen ganz oben, drücken Sie <RETURN>.

Wenn der Text erfolgreich geladen wurde, erscheint das Hauptmenü wieder auf dem Bildschirm. Entfernen Sie Ihre Textdiskette und legen Sie die Diskette ein, auf der alle Ihre Giga-Publish-Dateien gespeichert sind.

Mit <b> **speichern** wird Ihr Text auf diese Diskette gespeichert: Er liegt nach der Konvertierung in dem Format vor, das Giga-Publish verarbeiten kann.

Die Handhabung des Gesamtpakets erfordert sehr viel Übung. Wir wollen Ihnen hier einige Tips geben, die die Arbeit vereinfachen:

1. Überlegen Sie sich als erstes eine Skizze, in der Sie das Layout festlegen: Spaltenzahl, Anzahl der Text- und Bildboxen, Plazierung der Bilder, verwendete Zeichensätze etc.
2. Verwenden Sie Bilder, laden Sie den Bild-Konverter und konvertieren Sie die Grafiken. Notieren Sie sich die Werte für Druckbreite und Druckhöhe (Unterpunkt: Info ausgeben), die Sie für die Bildboxen benötigen.
3. Laden Sie Giga-Publish und wählen Sie »a) Layout«, dann »bearbeiten«. Entwerfen Sie jetzt Ihr gewünschtes Layout. Sind alle Boxen festgelegt, notieren Sie sich die zugehörigen Nummern. Dann verlassen Sie diesen Punkt.
4. Wählen Sie im Menü »Layout« nun »e) Extras« an. Dort stellen Sie die verschiedenen Parameter ein.
5. Vom Layout-Menü wählen Sie anschließend über Punkt <d> die gewünschten Zeichensätze aus.
6. Kehren Sie jetzt ins Hauptmenü zurück. Dort aktivieren Sie den Text-Editor. Schreiben Sie einen neuen Text oder laden Sie einen bestehenden; nicht vergessen, Mastertext muß vorher konvertiert worden sein. Denken Sie daran, die Steuerzeichen für die Wahl der Zeichensätze für jede Textbox anzugeben.
7. Nach Verlassen des Editors läßt sich der Text über Menüpunkt <c> ausdrucken. Achten Sie darauf, daß das Papier richtig eingelegt ist, um einen unsauberen Schriftausdruck zu vermeiden. Eventuell lösen Sie vor dem Ausdruck noch einen Zeilenvorschub beim Drucker aus.

(Dieter Bayer/ef)


```
2200 : e8 e9 ee ef ea eb bf 22  e8
2208 : 8f 22 19 22 35 22 5e 22  55
2210 : 52 22 78 22 6b 22 4c d1  72
2218 : 21 a4 d3 cc cc 22 f0 f6  a9
2220 : c6 d3 b1 d1 88 91 d1 c8  64
2228 : c8 cc cd 22 90 f4 88 a9  34
2230 : 7f 91 d1 d0 e1 ac cd 22  05
2238 : 88 b1 d1 c9 7f d0 d7 c4  ae
2240 : d3 f0 d3 88 b1 d1 c8 91  81
2248 : d1 88 c4 d3 d0 f5 a9 20  ac
2250 : d0 df a6 d3 ec cc 22 f0  d3
2258 : bd c6 d3 4c d1 21 a6 d3  5f
2260 : e8 ec cd 22 b0 b0 e6 d3  4a
2268 : 4c d1 21 ac cc 22 a9 7f  fe
2270 : 91 d1 c8 cc cd 22 d0 f8  d9
2278 : ad cc 22 85 d3 4c d1 21  ee
2280 : a4 d3 91 d1 c8 cc cd 22  1b
2288 : f0 02 84 d3 4c d1 21 ac  46
2290 : cd 22 88 b1 d1 c9 7f d0  d2
2298 : 07 cc cc 22 f0 21 d0 f2  be
22a0 : c8 8c cb 22 a2 00 ac cc  5c
22a8 : 22 b1 d1 2c ce 22 10 03  e1
22b0 : 20 cf 22 9d 00 02 e8 c8  39
22b8 : cc cb 22 d0 ec f0 02 a2  b0
22c0 : 00 8e cb 22 20 d3 24 ad  cb
22c8 : cb 22 60 00 00 00 00 c9  50
22d0 : 20 90 0f c9 40 90 10 c9  92
22d8 : 60 90 0a c9 80 90 03 a9  28
22e0 : 2e 60 09 40 2c 09 80 60  57
22e8 : 68 8d 20 23 68 8d 21 23  41
22f0 : 20 17 23 a8 30 19 20 17  f4
22f8 : 23 aa 20 3c 21 20 17 23  b6
2300 : c9 fe f0 1f aa 29 7f 91  7d
2308 : d1 c8 8a 10 f0 30 e1 ad  55
2310 : 21 23 48 ad 20 23 48 ee  a5
2318 : 20 23 d0 03 ee 21 23 ad  3e
2320 : ff ff 60 a2 27 a9 7e 9d  80
2328 : 00 04 9d a0 04 ca 10 f7  6c
2330 : 30 cb c9 20 90 18 c9 40  2e
2338 : 90 25 c9 60 90 17 c9 80  c3
2340 : 90 0f c9 c0 90 08 c9 e0  14
2348 : 90 0f c9 ff f0 0f a9 7f  ff
2350 : 60 38 e9 20 60 38 e9 40  3b
2358 : 60 38 e9 80 60 a9 5e 60  ec
2360 : d8 a9 00 8d 8d 02 a0 40  ab
2368 : 84 cb 8d 00 dc ae 01 dc  36
2370 : e0 ff d0 03 4c fd 23 a8  77
2378 : a9 81 85 f5 a9 eb 85 f6  00
2380 : a9 fe 8d 00 dc a2 08 48  9f
2388 : ad 01 dc cd 01 dc d0 f8  d3
2390 : 4a b0 16 48 b1 f5 c9 05  bd
2398 : b0 0c c9 03 f0 08 0d 8d  c0
23a0 : 02 8d 8d 02 10 02 84 cb  c7
23a8 : 68 c8 c0 41 b0 0b ca d0  fd
23b0 : df 38 68 2a 8d 00 dc d0  f9
23b8 : cc 68 4c 24 24 a4 cb b1  4a
23c0 : f5 aa c4 c5 f0 07 a0 10  de
23c8 : 8c 8c 02 d0 30 2c 8a 02  c8
23d0 : 30 12 70 45 c9 20 f0 0c  48
23d8 : c0 00 f0 08 c0 02 f0 04  bd
23e0 : c0 07 d0 35 ac 8c 02 f0  18
23e8 : 05 ce 8c 02 d0 2b ce 8b  70
23f0 : 02 d0 26 a0 04 8c 8b 02  cf
23f8 : a4 c6 88 10 1c a4 cb 84  43
2400 : c5 ac 8d 02 8c 8e 02 e0  c6
2408 : ff f0 0e 8a a6 c6 ec 89  bc
2410 : 02 b0 06 9d 77 02 e8 86  d8
2418 : c6 a9 7f 8d 00 dc ad 0d  fc
2420 : dc 4c 81 ea ad 8d 02 29  82
2428 : 07 0a aa bd 55 24 85 f5  0f
2430 : bd 56 24 85 f6 4c bd 23  e1
2438 : a5 c6 f0 17 78 ac 77 02  2e
2440 : a2 00 bd 78 02 9d 77 02  50
2448 : e8 e4 c6 d0 f5 c6 c6 98  50
2450 : 58 18 24 38 60 fc 1e 3d  a5
2458 : 1f 7e 1f fc 1e bf 1f fc  74
2460 : 1e fc 1e fc 1e 20 38 24  30
2468 : b0 fb c9 f1 f0 04 c9 0a  31
2470 : 18 60 a9 01 38 60 a9 00  70
2478 : 85 c6 a5 c6 f0 fc a9 00  40
2480 : 85 c6 60 a9 10 8d d2 24  b7
2488 : a9 0b 8d f8 07 a9 01 8d  16
2490 : 27 d0 a2 01 8e 15 d0 ca  52
2498 : 8e 1b d0 8a a2 3f 9d c0  55
24a0 : 02 ca 10 fa 8e c0 02 8e  7f
24a8 : c3 02 78 a9 b9 8d 14 03  1e
24b0 : a9 24 8d 15 03 58 4c e5  61
24b8 : 24 d8 ce d2 24 d0 0d a9  a7
24c0 : 10 8d d2 24 ad 15 d0 49  29
24c8 : 01 8d 15 d0 20 e5 24 4c  4a
24d0 : 60 23 00 a9 00 8d 15 d0  59
24d8 : 78 a9 60 8d 14 03 a9 23  35
24e0 : 8d 15 03 58 60 a5 d3 0a  5a
24e8 : 0a 0a a8 a9 00 2a aa 18  83
24f0 : 98 69 18 a8 8a 69 00 8d  67
24f8 : 10 d0 8c 00 d0 a5 d6 0a  3d
2500 : 0a 0a 18 69 39 8d 01 d0  e8
2508 : 60 53 3a 54 29 31 32 33  76
2510 : 34 35 36 37 38 39 30 31  c4
2518 : 32 33 34 ad 03 20 a2 72  47
2520 : a0 27 20 bd ff a9 01 a2  aa
2528 : 08 a0 02 20 ba ff 20 c0  b3
2530 : ff a2 01 4c c6 ff 18 ad  72
```

Listing 2. »tc 0« bitte mit dem MSE eingeben

```
2538 : 03 20 69 02 a2 0b a0 25   35
2540 : 20 bd ff a9 01 a2 08 a0   fa
2548 : 01 20 ba ff 20 c0 ff a2   55
2550 : 01 4c c9 ff 18 ad 03 20   25
2558 : 69 04 a2 09 a0 25 20 bd   bc
2560 : ff a9 01 a2 08 a0 0f 20   ca
2568 : ba ff 20 c0 ff a9 01 20   d3
2570 : c3 ff 4c 90 25 20 7b 25   e4
2578 : 4c 90 25 20 cc ff a9 01   cf
2580 : 4c c3 ff 20 a5 ff a6 90   c8
2588 : 60 20 a8 ff a6 90 60 00   93
2590 : 20 ef 25 a9 00 85 90 a9   e8
2598 : 0f a2 08 a0 6f 20 ba ff   f1
25a0 : a9 00 20 bd ff 20 c0 ff   0d
25a8 : a9 08 20 b4 ff a9 6f 20   3f
25b0 : 96 ff a2 00 20 a5 ff 9d   59
25b8 : 00 02 e8 24 90 50 f5 a9   2f
25c0 : 08 20 ab ff a9 0f 20 c3   de
25c8 : ff ad 8f 25 d0 08 ad 00   2a
25d0 : 02 c9 30 d0 01 60 a2 00   7b
25d8 : bd 00 02 c9 0d f0 08 9d   03
25e0 : 99 07 e8 e0 26 d0 f1 a9   57
25e8 : 00 85 c6 a5 c6 f0 fc a9   4c
25f0 : 00 85 c6 a2 27 a9 7f 9d   b2
25f8 : 98 07 ca 10 fa ad 00 02   ea
2600 : c9 30 60 20 9e 27 b0 fa   d9
2608 : 8d 03 20 a9 00 85 50 a9   15
2610 : 40 85 51 20 1b 25 20 83   ce
2618 : 25 d0 23 c9 00 d0 31 20   33
2620 : 83 25 d0 1a c9 08 d0 28   1e
2628 : ad 81 27 c9 54 f0 04 c9   09
2630 : 4b d0 1d 8d 31 2b a2 12   f7
2638 : 20 ed 28 20 4b 2a 20 75   ce
2640 : 25 f0 01 60 a0 00 a9 ff   da
2648 : 91 50 a9 01 8d 05 20 60   ce
2650 : 20 75 25 a2 13 20 ed 28   03
2658 : 20 e8 22 01 15 4b 05 09   67
2660 : 0e 05 7f 4d 01 13 14 05   7d
2668 : 12 14 05 18 14 2d 44 01   86
2670 : 14 05 89 ff 4c 76 24 ad   cd
2678 : 05 20 d0 25 20 e8 22 01   3a
2680 : 15 4b 05 09 0e 7f 54 05   d6
2688 : 18 14 7f 09 0d 7f 53 10   e6
2690 : 05 09 03 08 05 92 ff 4c   59
2698 : 76 24 60 20 89 25 4c 75   1a
26a0 : 25 20 e8 22 01 15 54 05   68
26a8 : 18 14 0e 01 0d 05 ba ff   52
26b0 : a0 15 a2 0a a9 8e 20 a7   a4
26b8 : 21 f0 0e a2 00 bd 00 02   1b
26c0 : 9d 0d 25 e8 ec cb 22 d0   a2
26c8 : f4 ad cb 22 8d 03 20 f0   1d
26d0 : c9 a9 00 85 50 a9 40 85   7d
26d8 : 51 a2 ff 8e 09 20 20 36   ca
26e0 : 25 20 e8 22 01 17 54 05   b8
26e8 : 18 14 7f 17 09 12 04 7f   fd
26f0 : 07 05 13 10 05 09 03 08   f5
26f8 : 05 12 94 ff a0 00 b1 50   9d
2700 : c9 ff f0 97 20 51 27 a5   6c
2708 : 90 d0 09 e6 50 d0 ed e6   31
2710 : 51 4c fc 26 20 75 25 ad   29
2718 : 00 02 c9 36 f0 03 4c 77   9a
2720 : 26 ad 01 02 c9 33 f0 03   9d
2728 : 4c 77 26 20 e8 22 01 17   8f
2730 : 54 05 18 14 7f 5d 02 05   84
2738 : 12 13 03 08 12 05 09 02   07
2740 : 05 0e 7f bf ff 20 65 24   03
2748 : f0 01 60 20 54 25 4c d1   18
2750 : 26 c9 20 d0 06 cd 09 20   b0
2758 : d0 01 60 8d 09 20 4c a8   87
2760 : ff 00 00 00 00 00 00 00   60
2768 : 00 00 00 00 00 00 00 00   69
2770 : 00 00 00 00 00 00 00 00   71
2778 : 00 00 00 00 00 00 00 00   79
2780 : 00 00 24 30 a9 02 a2 82   ca
2788 : a0 27 20 bd ff a9 01 a2   12
2790 : 08 a0 00 20 ba ff 20 c0   9a
2798 : ff a2 01 4c c6 ff 20 84   a8
27a0 : 27 20 83 25 d0 6f c9 22   51
27a8 : d0 f7 a0 00 20 83 25 99   82
27b0 : 61 27 d0 61 c8 c9 22 d0   0a
27b8 : f3 88 c0 10 f0 05 a9 00   ff
27c0 : 99 61 27 a9 00 85 02 a9   90
27c8 : 00 85 50 a9 17 85 51 20   f7
27d0 : 83 25 d0 41 c9 00 d0 f7   12
27d8 : a0 03 20 83 25 d0 36 88   35
27e0 : 10 f8 20 83 25 d0 2e c9   0a
27e8 : 22 d0 f7 a0 00 20 83 25   de
27f0 : 91 50 d0 21 c8 c9 22 d0   07
27f8 : f4 88 c0 10 f0 04 a9 00   38
2800 : 91 50 18 a5 50 69 10 85   10
2808 : 50 90 02 e6 51 e6 02 a5   9d
2810 : 02 c9 90 90 ba 20 75 25   fa
2818 : f0 02 38 60 a5 02 f0 fa   48
2820 : 85 03 20 ff 28 a0 0f a9   46
2828 : 7e 99 22 05 88 10 fa a0   d2
2830 : 00 b9 61 27 f0 0b 20 32   96
2838 : 23 99 fa 04 c8 c0 10 d0   db
2840 : f0 a9 3e 8d 49 05 a9 3c   22
2848 : 8d 5a 05 a9 00 85 02 a9   00
2850 : 00 85 50 a9 17 85 51 20   7f
2858 : a9 28 20 38 24 b0 fb c9   70
2860 : f1 d0 02 38 60 c9 ec d0   eb
2868 : 17 a6 02 e8 e4 03 f0 ea   70
2870 : 86 02 18 a5 50 69 10 85   4e
2878 : 50 90 02 e6 51 4c 57 28   93
2880 : c9 ed d0 17 a6 02 ca 30   5d
2888 : d1 86 02 38 a5 50 e9 10   c9
2890 : 85 50 a5 51 e9 00 85 51   28
2898 : 4c 57 28 c9 80 d0 03 4c   06
28a0 : 13 29 c9 20 d0 b4 4c 9e   df
28a8 : 27 a5 50 85 52 a5 51 85   09
28b0 : 53 a5 02 85 04 a2 08 a0   be
28b8 : 00 20 03 29 b1 52 f0 11   42
28c0 : 20 32 23 91 d1 c8 c0 10   7b
28c8 : d0 f2 f0 09 a9 7f 91 d1   ef
28d0 : c8 c0 10 d0 f7 e8 e0 19   93
28d8 : f0 24 18 a5 52 69 10 85   51
28e0 : 52 90 02 e6 53 e6 04 a5   9f
28e8 : 04 c5 03 90 ca 20 3c 21   83
28f0 : a0 27 a9 7f 91 d1 88 10   68
28f8 : fb e8 e0 19 d0 ef 60 a2   16
2900 : 06 d0 ea 18 bd 75 21 69   0b
2908 : 0a 85 d1 bd 8e 21 69 00   98
2910 : 85 d2 60 a0 00 b1 50 f0   db
2918 : 08 99 72 27 c8 c0 10 d0   e3
2920 : f4 98 18 60 a9 02 a2 82   ad
2928 : a0 27 20 8a 27 20 ff 28   79
2930 : a9 06 85 d3 a9 06 85 d6   47
2938 : 20 83 25 d0 37 20 83 25   4a
2940 : 20 83 25 d0 2f 20 83 25   d2
2948 : d0 2a 20 83 25 d0 25 85   1e
2950 : 02 20 83 25 d0 1e 85 03   02
2958 : 20 b3 29 20 83 25 d0 14   6d
2960 : c9 00 f0 06 20 82 29 4c   7a
2968 : 5b 29 a9 80 20 82 29 b0   ee
2970 : 0e 4c 40 29 a5 d3 05 d6   94
2978 : c9 06 f0 03 20 db 29 4c   ff
2980 : 75 25 c9 80 f0 11 20 32   87
2988 : 23 c9 20 d0 02 a9 7f a4   67
2990 : d3 91 d1 e6 d3 18 60 a9   50
2998 : 06 85 d3 a5 d6 c9 17 f0   04
29a0 : 04 e6 d6 18 60 20 db 29   99
29a8 : b0 08 20 ff 28 a9 06 85   57
29b0 : d6 18 60 20 3a 21 a6 02   fa
29b8 : a5 03 20 ba 20 a2 00 a4   9f
29c0 : d3 bd 00 02 c9 7f d0 05   98
29c8 : e8 e0 04 d0 f4 bd 00 02   7d
29d0 : 91 d1 c8 e8 e0 05 d0 f5   fe
29d8 : 84 d3 60 20 e8 22 22 18   ba
29e0 : 54 01 13 14 85 ff a9 00   fb
29e8 : 85 c6 20 38 24 b0 fb c9   2b
29f0 : f1 f0 02 18 60 38 60 a9   7a
29f8 : 3e 8d 49 07 8d 8f 25 a9   6d
2a00 : a4 a2 02 a0 15 20 a7 21   bd
2a08 : f0 3b ae 00 02 e0 24 d0   9b
2a10 : 0a a2 00 8e 8f 25 a0 02   e6
2a18 : 4c 2a 29 a9 00 20 bd ff   f1
2a20 : a9 01 a2 08 a0 0f 20 ba   6c
2a28 : ff 20 c0 ff a2 01 20 c9   ae
2a30 : ff a2 00 bd 00 02 20 a8   1a
2a38 : ff e8 ec cb 22 d0 f4 20   1d
2a40 : 75 25 4c f7 29 a9 00 8d   55
2a48 : 8f 25 60 20 e8 22 01 15   54
2a50 : 4d 01 13 14 05 12 14 05   a0
2a58 : 18 14 2d 44 01 14 05 09   25
2a60 : 7f 0c 01 04 05 8e ff a9   be
2a68 : 00 8d 30 2b 20 fc 2a c9   c6
2a70 : 00 f0 21 20 9e 2a 90 08   c2
2a78 : a8 b9 0a 20 c9 ff f0 ec   be
2a80 : a0 00 91 50 a6 90 d0 14   e9
2a88 : e6 50 d0 02 e6 51 a5 51   3d
2a90 : c9 d0 d0 d8 a0 00 a9 ff   c1
2a98 : 91 50 18 60 38 60 c9 d3   b9
2aa0 : f0 02 38 60 20 fc 2a c9   d2
2aa8 : 2d d0 09 20 fc 2a 29 01   4b
2ab0 : 18 69 8b 60 c9 06 d0 09   8e
2ab8 : 20 fc 2a 29 01 18 69 88   8e
2ac0 : 60 c9 0f d0 04 a9 92 18   eb
2ac8 : 60 c9 15 d0 04 a9 93 18   78
2ad0 : 60 c9 0e d0 04 a9 91 18   b6
2ad8 : 60 c9 21 f0 1b c9 3a b0   ce
2ae0 : 04 c9 30 b0 13 c9 0b f0   79
2ae8 : 0c c9 12 f0 08 c9 13 f0   78
2af0 : 04 c9 1a d0 ad 20 fc 2a   9e
2af8 : 38 a9 00 60 20 a5 ff ae   9e
2b00 : 30 2b e8 e0 50 90 02 a2   f3
2b08 : 00 8e 30 2b c9 8e f0 01   98
2b10 : 60 ad 31 2b c9 4b f0 10   d3
2b18 : ad 30 2b c9 50 b0 09 20   d0
2b20 : a5 ff ee 30 2b 4c 18 2b   52
2b28 : a9 00 8d 30 2b a9 8e 60   36
2b30 : 00 00 a9 0b 8d 11 d0 78   92
2b38 : a9 60 8d 14 03 a9 23 8d   1c
2b40 : 15 03 58 a9 36 85 01 20   f6
2b48 : 47 21 a9 12 8d 18 d0 a9   fd
2b50 : 00 85 c6 8d 05 20 8d 8f   1d
2b58 : 25 a9 80 8d 8a 02 a9 1b   b9
2b60 : 8d 11 d0 20 47 21 20 e8   7e
2b68 : 22 01 02 fe 54 05 18 14   61
2b70 : 2d 4b 0f 0e 16 05 12 14   c2
2b78 : 05 92 02 06 01 29 7f 0c   77
2b80 : 01 04 05 8e 02 08 02 29   51
2b88 : 7f 13 10 05 09 03 08 05   08
2b90 : 12 8e 02 0a 18 29 7f 45   ff
2b98 : 18 09 94 ff 20 38 24 a2   f3
2ba0 : 05 dd cf 2b f0 05 ca 10   70
2ba8 : f8 30 f1 8a a8 0a aa bd   87
2bb0 : db 2b 8d ca 2b bd dc 2b   48
2bb8 : 8d cb 2b be d5 2b f0 09   5a
2bc0 : 20 3c 21 a0 01 a9 73 91   a9
2bc8 : d1 20 ff ff 4c 63 2b 01   38
2bd0 : 02 18 e0 e2 e4 06 08 0a   25
2bd8 : 00 00 00 03 26 77 26 e7   bf
2be0 : 2b 24 29 f7 29 fe 2b ee   7c
2be8 : 20 d0 a9 00 85 c6 20 38   5a
2bf0 : 24 b0 fb c9 80 f0 04 ce   e2
2bf8 : 20 d0 60 4c e2 fc ee 8f   13
2c00 : 25 20 90 25 a9 00 8d 8f   ee
2c08 : 25 60 ff 00 ff 00 ff 00   5d
```

Listing 2. (Schluß)

Der C 64 als Synthesizer

Jetzt kommen die Sound-Eigenschaften des C 64 ans Tageslicht. Die Computertasten werden zum Keyboard mit einstellbaren Klangeigenschaften – und alles ist am Monitor zu sehen.



Bild 1. Der Bildschirm zeigt alle Werte auf einem Blick

Haben Sie sich nicht schon immer eine kleine Heimorgel gewünscht? Mit dem Programm »Synthesizer« verwandeln Sie Ihren C 64 in dieses Instrument (Bild 1). Drücken Sie auf die Tasten und lassen Sie sich überraschen.

Um die Möglichkeiten zu nutzen, sind folgende Parameter einstellbar und werden übersichtlich – teilweise grafisch unterstützt – auf dem Bildschirm dargestellt.

1) Die Tastenreihe »Q« bis »RETURN« bildet die Orgeltastatur. Sie umfaßt zwei Oktaven mit allen Halbtönen und erscheint oben auf dem Bildschirm. Beim Drücken einer Taste erklingt der Ton so lange, bis sie wieder losgelassen wird. Gleichzeitig zeigt ein gelber Balken die aktivierte Orgeltaste an.

2) Anschlag, Abschwellen, Haltepegel und Ausklingen der Orgeltasten sind mit den Funktionstasten einstellbar. Die geSHIFTeten Tasten verringern, die ungeSHIFTeten erhöhen den Wert. Die relative Lautstärke (Pegel) des Tones wird ständig rechts oben als Balken auf dem Bildschirm gezeigt.

3) Als Wellenformen kann man Dreieck, Sägezahn, Rechteck, Rauschen und die ringmodulierte Dreieckschwingung (RING) mit der Taste <Z> wählen. Die beiden letzten sind in Klammern eingefaßt, da sie sich zum Musizieren nicht eignen.

4) Wählt man die Rechteckschwingung, wird das Tastverhältnis angezeigt und kann mit den Tasten <N> und

Kurzinfo: Synthesizer

Programmart: Musikprogramm
Laden mit: LOAD "SYNTHESIZER",8
Starten mit: Nach dem Laden RUN eingeben.
Eingaben über: Tastatur
Besonderheiten:
Die Maschinenroutine »SYNTHESIZER-OBJ« wird nach dem Starten automatisch nachgeladen.
Programmautor: Martin Ahlborn

<M> in Sechzehntel-Abstufungen verändert werden. Das Low-Byte des Tastverhältnisses beträgt immer 128.
5) Der Synthesizer nutzt das im Anhang P des Handbuchs gezeigte Frequenzspektrum voll aus: Der tiefste Ton ist der Halbton unter C_0 und der höchste ist das Ais-7 (!).
Da die Tastatur aber nur zwei Oktaven umfaßt, kann man mit den Cursortasten den Frequenzbereich um jeweils eine Oktave ändern.
6) Mit der Taste <C> wird der Filtermodus umgeschaltet, wobei das entsprechende Wort auf dem Bildschirm revers geschrieben wird. Ist kein Filtermodus eingeschaltet, werden die Schaltbits 0,1 und 2 im Register 23 gelöscht, also für alle drei Stimmen die Filter ausgeschaltet. Auch die Verwendung mehrerer Filtermodi ist zugelassen.
7) Die folgenden Parameter sind nur bei eingeschalteten Filtern hör- und sichtbar:
7a) Mit den (ungeshifteten) Tasten <[> und <]> verändert man die Grenzfrequenz der Filter in Sechzehntelschritten. Es werden also die höchstwertigen 4 Bit im Register 22 variiert.
7b) Die Filterresonanz wird mit den Tasten <<> und <>> vergrößert oder verkleinert.
7c) Die Tasten <?> beziehungsweise <.> schalten den »Wah-wah«-Effekt (er heißt wie er klingt) ein und aus. Nach dem Einschalten wird die Filtergrenzfrequenz (Register 22) gemäß dem Verlauf der Hüllkurve (Register 28) verändert. Die eingestellte Filterfrequenz (siehe 7a) hat dann keine Bedeutung.
8) Mit der Stopp-Taste verläßt man das Programm. Da es jetzt im Speicher ist, kann es mit RUN 1000 unverzüglich wieder gestartet werden. Nun haben alle Parameter wieder die ursprünglichen Werte.
Ich möchte nicht verheimlichen, daß der »Synthesizer« natürlich auch seine Grenzen hat. Folgende Punkte sind zu beachten:
1) Alle Parameter werden jeweils für alle drei Stimmen gleichzeitig eingestellt, denn bei nur einem Manual wäre es sinnlos, den verschiedenen Stimmen zum Beispiel verschiedene Wellenformen zu geben.
2) Es gibt natürlich grenzenlos viele Möglichkeiten, Parameter (zum Beispiel die Filterfrequenz) während des Erklingens eines Tones zu verändern. Ihr Einbau würde aber ebenfalls den Rahmen dieses Programms sprengen.
3) Die Lautstärke ist fest auf 15 eingestellt. (Wozu hat man schließlich den Lautstärke-Regler am Fernseher?)
In Tabelle 1 finden Sie übrigens einige Beispiele dafür, was man mit dem Programm alles machen kann.

(Martin Ahlborn/kn)

Charakter	Okt.	Wellenf.	Tastv.	An	Ab	Hlt	Aus	Filt	Wah	FF	Res
E-Gitarre	4-6	Rechteck	1	0	9	0	9	HBT	Ein	-	7
Mandoline	5-7	Rechteck	C	0	8	00	8	-	-	-	-
Mundharmonika	6-8	Sägezahn	-	8	0	F	7	-	-	-	-
Gong	3-5	Ring	-	0	D	0	D	-	-	-	-
Explosion	0-2	Rauschen	-	0	0	F	C	-	-	-	-
Popcorn	5-7	Dreieck	-	0	2	0	0	-	-	-	-
Quak-quak	4-6	Sägezahn	-	8	9	9	8	B	Ein	-	F
Meeresrauschen	5-7	Rauschen	-	B	B	0	B	HBT	Ein	-	7
Knattern	0-2	Rechteck	0	8	0	F	D	-	-	-	-
Xylophon	4-6	Dreieck	-	0	9	0	9	T	Aus	2	6
Glöckchen	6-8	Dreieck	-	0	A	0	A	-	-	-	-

Tabelle 1. Klangeinstellungen mit »Synthesizer«


```
1 IF A=0 THEN A=A+1:LOAD"SYNTHESIZER-OBJ",
  8,1                                      <219>
2 REM -------  C-64 SYNTHESIZER  -------   <095>
3 REM   VON MARTIN AHLBORN                 <177>
5 REM                                      <067>
990 REM ***************                    <116>
992 REM *TEXT & GRAFIK*                    <029>
994 REM ***************                    <120>
996 :                                      <210>
1000 PRINT CHR$(142)"{CLR,CTRL-H,2RIGHT,PU
     RPLE}********{2SPACE}C-64 SYNTHESIZER
     {2SPACE}********{WHITE}"              <253>
1010 PRINT"{DOWN,SPACE,RVSON,SPACE,RIGHT,S
     PACE,RIGHT,SPACE}_{SPACE,RIGHT,SPACE,
     RIGHT,SPACE,RIGHT,SPACE}_{SPACE,RIGHT
     ,SPACE,RIGHT,SPACE}_{SPACE,RIGHT,SPAC
     E,RIGHT,SPACE,RIGHT,SPACE}_{SPACE,RVO
     FF,3SPACE}MAX{SPACE,RVSON}Y"          <207>
1020 PRINT"{SPACE,RVSON,SPACE,RIGHT,SPACE,
     RIGHT,SPACE}_{SPACE,RIGHT,SPACE,RIGHT
     ,SPACE,RIGHT,SPACE}_{SPACE,RIGHT,SPAC
     E,RIGHT,SPACE}_{SPACE,RIGHT,SPACE,RIG
     HT,SPACE,RIGHT,SPACE}_{SPACE,7RIGHT,S
     PACE}"                                <130>
1030 PRINT"{SPACE,RVSON,SPACE,RIGHT,SPACE,
     RIGHT,SPACE}_{SPACE,RIGHT,SPACE,RIGHT
     ,SPACE,RIGHT,SPACE}_{SPACE,RIGHT,SPAC
     E,RIGHT,SPACE}_{SPACE,RIGHT,SPACE,RIG
     HT,SPACE,RIGHT,SPACE}_R{RVOFF,SPACE}P
     EGEL{SPACE,RVSON,SPACE}"              <109>
1040 PRINT"{SPACE,RVSON,SPACE}_ _ _ _ _
     _ _ _ _ _ _ _ _E{7RIGHT,SPACE}"       <040>
1050 PRINT"{SPACE,RVSON}Q_W_E_R_T_Y_U_I_O_
     P_@_*_↑_=_T{RVOFF,3SPACE}MIN{SPACE,RV
     SON}T"                                <028>
1060 PRINT"{2DOWN,RIGHT,GREY 2}TASTEN{4SPA
     CE}PARAMETER{4SPACE}0123456789ABCDEF"
     ;                                     <118>
1070 PRINT" TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
     TTTTTTTTT";                           <064>
1080 PRINT"{ORANGE,SPACE}F1{2SPACE}F2{4SPA
     CE}ANSCHLAG{5SPACE,RVSON}LLLLLL{RVOFF
     ,10SPACE}";                           <065>
1090 PRINT"{GREEN,SPACE}F3{2SPACE}F4{4SPAC
     E}ABSCHWELLEN{2SPACE,RVSON}L{RVOFF,15
     SPACE}";                              <123>
1100 PRINT"{LIG.BLUE,SPACE}F5{2SPACE}F6{4S
     PACE}HALTEN{7SPACE,RVSON}LLLLLLLLLLLL
     LLLL{RVOFF}";                         <028>
1110 PRINT"{YELLOW,SPACE}F7{2SPACE}F8{4SPA
     CE}AUSKLINGEN{3SPACE,RVSON}LLLLLLL{RV
     OFF,9SPACE}";                         <034>
1120 PRINT"{BLACK,SPACE}'N' 'M'{3SPACE}TAS
     TVERH.{4SPACE,RVSON}LLLLLL{RVOFF,10SP
     ACE}";                                <050>
1130 PRINT"{LIG.GREEN,2SPACE}'Z'{6SPACE}WE
     LLENFORM{3SPACE}DREIECK{3SPACE}"      <229>
1140 PRINT"{PURPLE,2SPACE}CRSR{5SPACE}OKTA
     VEN{6SPACE}C-{WHITE}4{PURPLE,SPACE}BI
     S C-{WHITE}6"                         <088>
1150 PRINT"{GREY 3,2SPACE}'C'{6SPACE}FILTE
     R{7SPACE}HOCH BAND TIEF"              <099>
1160 PRINT"{BLACK,SPACE}'[' ']'{3SPACE}FIL
     TERFREQ.{2SPACE,RVSON}LLLLLLL{RVOFF,9
     SPACE}";                              <072>
1170 PRINT"{RVOFF,SPACE}'<' '>'{3SPACE}RES
     ONANZ{5SPACE,RVSON}LLLLLLL{RVOFF,9SPA
     CE}";                                 <210>
1180 PRINT"{RVOFF,2SPACE}'?'{6SPACE}'WAH-W
     AH'{4SPACE}AUS"                       <144>
1190 PRINT"{RED,SPACE}RUN/STOP{2SPACE}ENDE
     "                                     <055>
1200 PRINT"{DOWN,GREY 1,8SPACE}MARTIN AHLB
     ORN 3418 USLAR{UP}"                   <204>
1300 :                                     <006>
1305 REM ************                      <032>
1310 ::::::SYS 12800::                     <179>
1320 REM ************                      <047>
2000 END E                                 <010>
```

Listing 1. »SYNTHESIZER« – das Hauptprogramm mit dem Bildschirmaufbau.
Bitte mit dem Checksummer (Seite 159) eingeben.

```
Name : synthesizer-obj   2f00 35de
----------------------------------
2f00 : 05 01 15 01 26 01 38 01  39
2f08 : 4b 01 5f 01 74 01 8a 01  47
2f10 : a1 01 ba 01 d4 01 f0 01  1c
2f18 : 0e 02 2d 02 4e 02 71 02  72
2f20 : 96 02 bd 02 e7 02 13 03  48
2f28 : 42 03 74 03 a9 03 e1 03  aa
2f30 : 1c 04 5b 04 9d 04 e3 04  37
2f38 : 2d 05 7c 05 d0 05 28 06  89
2f40 : 86 06 e9 06 52 07 c1 07  77
2f48 : 37 08 b4 08 38 09 c4 09  a2
2f50 : 59 0a f6 0a 9d 0b 4e 0c  31
2f58 : 09 0d cf 0d a1 0e 80 0f  28
2f60 : 6c 10 66 11 6f 12 88 13  60
2f68 : b1 14 ec 15 3a 17 9c 18  00
2f70 : 13 1a a0 1b 45 1d 03 1f  a3
2f78 : db 20 cf 22 e1 24 12 27  71
2f80 : 65 29 db 2b 77 2e 3a 31  0a
2f88 : 27 34 41 37 8a 3a 05 3e  0c
2f90 : b5 41 9d 45 c1 49 24 4e  89
2f98 : ca 52 b6 57 ed 5c 74 62  7c
2fa0 : 4f 68 83 6e 15 75 0b 7c  f4
2fa8 : 6b 83 3c 8b 83 93 49 9c  89
2fb0 : 94 a5 6c af da b9 e7 c4  cc
2fb8 : 9c d0 04 dd 28 ea 14 f8  95
2fc0 : 34 29 3a 97 31 39 38 2c  20
2fc8 : 30 00 cf 14 ed 03 3a 00  4f
2fd0 : ed 14 ee 03 99 22 20 22  53
2fd8 : 3b 54 24 3a 99 22 11 11  9f
2fe0 : 91 20 3f 20 22 3b 46 24  b3
2fe8 : 3b 22 20 22 3b 00 07 15  7b
2ff0 : ef 03 58 b2 31 3a 8b 46  6d
2ff8 : 24 b3 b1 22 22 a7 58 b2  cd
3000 : 10 40 18 50 20 40 28 50  86
3008 : 30 40 38 50 48 50 50 40  39
3010 : 58 50 60 40 68 50 70 40  fc
3018 : 78 50 88 50 90 40 98 50  f3
3020 : a0 40 a8 50 b8 50 c0 40  a6
3028 : c8 50 d0 40 d8 50 e0 40  69
3030 : e8 50 f8 50 15 f3 03 3a  fa
3038 : 92 31 39 38 2c 31 3a a1  31
3040 : 7f 01 7f 40 7f 08 fd 02  5c
3048 : fd 01 fd 40 fb 02 fb 01  0f
3050 : fb 40 fb 08 f7 02 f7 01  dd
3058 : f7 40 ef 02 ef 01 ef 40  f3
3060 : ef 08 df 02 df 40 df 08  1b
3068 : bf 02 bf 01 bf 40 bf 08  45
3070 : bf 20 fe 02 31 30 00 85  df
3078 : 15 f6 03 3a 8b 4e b1 4c  9b
3080 : a7 31 30 31 31 00 94 15  82
3088 : f7 03 3a 8b 4d b2 30 a7  7b
3090 : 31 30 32 31 00 ac 15 f8  38
3098 : 03 3a 8b 46 24 b2 22 2c  1d
30a0 : 22 b0 46 24 b2 5a 24 a7  0e
30a8 : 31 30 31 31 00 c7 15 f9  ea
30b0 : 03 28 12 01 15 13 03 08  72
30b8 : 05 0e 29 12 05 03 08 14  02
30c0 : 05 03 0b 20 20 13 01 05  b6
30c8 : 07 05 1a 01 08 0e 20 04  72
30d0 : 12 05 09 05 03 0b 20 20  91
30d8 : 20 28 12 09 0e 07 29 20  b0
30e0 : 20 20 20 05 09 0e 01 15  e8
30e8 : 13 46 24 b3 22 41 22 b0  b4
30f0 : 46 24 b1 22 5a 22 a7 31  b1
30f8 : 30 31 31 00 01 16 fe 03  d0
3100 : 82 00 07 16 ff 03 3a 00  08
3108 : 1a 16 01 04 99 c7 28 32  cb
3110 : 30 29 3b 3a 97 32 31 31  1d
3118 : 2c 31 00 31 16 02 04 97  b4
3120 : 36 33 31 2c 31 33 3a 97  86
3128 : 31 39 38 2c 31 3a 85 46  11
3130 : 24 00 4c 16 06 04 8b 46  65
3138 : 24 b2 58 24 a7 99 22 91  43
3140 : 1d 1d 1d 1d 22 3b 3a 89  cf
3148 : 31 30 30 39 00 52 16 0c  c8
3150 : 04 3a 00 58 16 49 04 8e  55
3158 : 00 5e 16 4a 04 3a 00 64  31
3160 : 16 4b 04 3a 00 7b 16 d0  3a
3168 : 07 8f 22 20 44 41 54 45  ee
3170 : 49 20 41 55 46 4c 49 53  57
3178 : 54 45 4e 00 92 16 d2 07  36
3180 : 8f 22 20 c0 c0 c0 c0 c0  d7
3188 : c0 c0 c0 c0 c0 c0 c0 c0  87
3190 : c0 c0 00 98 16 d4 07 3a  5c
3198 : 00 a5 16 da 07 81 4e 52  a6
31a0 : b2 30 a4 46 47 00 bf 16  fc
31a8 : dc 07 3a 99 22 11 20 2d  4f
31b0 : 3e 12 22 3b 4b 57 24 28  38
31b8 : 4e 52 29 3b 22 92 22 00  20
31c0 : d3 16 de 07 3a 99 22 20  70
31c8 : 20 20 22 3b 46 57 24 28  e8
31d0 : 4e 52 29 00 e2 16 e0 07  02
31d8 : 3a 81 57 b2 30 a4 36 30  60
31e0 : 3a 82 00 fd 16 e1 07 3a  1c
31e8 : 92 31 39 38 2c 31 3a a1  e1
31f0 : 46 24 3a 99 22 91 91 20  3f
31f8 : 20 20 11 11 22 00 0f 17  1b
3200 : d8 a9 00 a2 7e 9d 80 03  de
3208 : ca 10 fa a9 ff a2 24 9d  af
3210 : c1 03 ca ca ca d0 f8 8d  91
3218 : 81 03 a2 03 a0 00 8c 20  a0
3220 : d0 8c 21 d0 8c e9 07 8c  e6
3228 : f4 07 8c 2a d0 a9 0e 8d  b6
3230 : fb 07 a2 08 8e 10 d0 ca  9b
3238 : 8e 27 d0 8e 28 d0 8e 29  f5
3240 : d0 8e 17 d0 a9 60 85 fa  e1
3248 : 8d f2 07 8d f1 07 8d 16  7c
3250 : d4 8d 17 d4 a9 34 8d f3  a6
3258 : 07 a9 02 8d 15 d4 a9 2f  63
3260 : 85 fb a9 80 8d 8a 02 8d  ae
3268 : 02 d4 8d 09 d4 8d 10 d4  fc
3270 : a2 10 8e ee 07 a9 39 8d  59
3278 : 06 d0 ca 8e f8 07 8e f9  61
3280 : 07 8e fa 07 8e 18 d4 8e  88
3288 : ef 07 a9 05 8d e8 07 8d  5d
3290 : ec 07 a9 0f 8d ea 07 a9  ec
3298 : 06 8d eb 07 20 54 34 ac  0f
32a0 : ee 07 8c 04 d4 8c 0b d4  3d
32a8 : 8c 12 d4 a9 08 8d 15 d0  8a
32b0 : 10 0b aa a9 cc 9d a8 05  8c
32b8 : a9 20 9d a9 05 ad 1c d4  e6
32c0 : ac f4 07 f0 03 8d 16 d4  65
32c8 : 4a 4a 4a 85 02 38 a9 64  cc
32d0 : e5 02 8d 07 d0 a6 cb e0  2e
32d8 : 40 f0 c4 78 a0 00 ba 86  d2
32e0 : 02 b9 40 30 8d 00 dc ad  7d
32e8 : 01 dc 39 41 30 d0 03 4c  fc
32f0 : 11 34 c8 c8 c0 34 d0 e9  2b
32f8 : ba e4 02 f0 03 4c 87 34  dc
3300 : 58 20 3e f1 c9 03 d0 03  34
3308 : 4c c3 35 c9 43 d0 03 4c  1c
3310 : 55 35 c9 3b d0 05 a2 00  99
3318 : 4c a5 33 c9 3a d0 05 a2  c0
3320 : 00 4c bc 33 c9 2e d0 05  37
3328 : a2 01 4c a5 33 c9 2c d0  e6
3330 : 05 a2 01 4c bc 33 c9 2f  3b
3338 : d0 03 4c 30 34 c9 1d d0  4b
3340 : 03 4c e7 33 c9 11 d0 03  38
3348 : 4c f7 33 c9 5a d0 03 4c  67
3350 : f3 34 c9 4d d0 04 a2 04  39
3358 : 10 1b c9 4e d0 04 a2 04  f2
3360 : 10 27 c9 85 90 04 c9 8d  92
3368 : 90 03 4c bd 32 38 e9 85  dc
3370 : c9 04 b0 11 aa bd e8 07  d4
3378 : c9 0f f0 ee 38 69 00 9d  ed
3380 : e8 07 4c 97 33 38 e9 04  96
3388 : aa bd e8 07 d0 03 4c bd  fe
3390 : 32 38 e9 01 9d e8 07 48  47
3398 : 20 54 34 68 18 69 28 ca  ff
33a0 : 10 fa 4c b2 32 bd f1 07  7e
33a8 : 29 f0 c9 f0 f0 36 bd f1  75
33b0 : 07 18 69 10 9d f1 07 9d  e0
33b8 : 16 d4 d0 13 bd f1 07 29  a9
33c0 : f0 f0 21 bd f1 07 38 e9  34
33c8 : 10 9d f1 07 9d 16 d4 4a  77
33d0 : 4a 4a 4a ca d0 03 18 69  84
33d8 : 28 aa a9 cc 9d 10 07 a9  23
33e0 : 20 9d 11 07 4c bd 32 a5  bb
33e8 : fa c9 90 f0 21 18 69 18  b2
33f0 : 85 fa ee f3 07 d0 0c a5  9f
33f8 : fa f0 13 38 e9 18 85 fa  a2
3400 : ce f3 07 ae f3 07 8e c2  97
3408 : 06 e8 e8 8e ca 06 4c bd  18
3410 : 32 ba 8a 18 69 0c c5 02  57
3418 : d0 03 4c f2 32 b9 00 30  2d
3420 : 48 b9 01 30 48 b1 fa 48  1a
3428 : c8 b1 fa 48 88 4c f2 32  ac
3430 : ad f4 07 a0 02 49 01 8d  b7
3438 : f4 07 f0 0c b9 e3 30 99  1c
3440 : 60 07 88 10 f7 4c bd 32  85
3448 : b9 e6 30 99 60 07 88 10  34
3450 : f7 4c bd 32 ad e8 07 0a  75
3458 : 0a 0a 0a 0d e9 07 8d 05  a3
3460 : d4 8d 0c d4 8d 13 d4 ad  b9
3468 : ea 07 0a 0a 0a 0a 0d eb  97
3470 : 07 8d 06 d4 8d 0d d4 8d  0a
3478 : 14 d4 ad ec 07 8d 03 d4  92
3480 : 8d 0a d4 8d 11 d4 60 ad  8e
3488 : f4 07 d0 06 ad f1 07 8d  96
3490 : 16 d4 ad 8d 0f d4 68 8d  30
3498 : 0e d4 ac ee 07 c8 8c 12  26
34a0 : d4 68 8d 01 d0 68 8d 00  b2
34a8 : d0 a9 09 8d 15 d0 ba e4  cd
34b0 : 02 f0 3d 68 8d 08 d4 68  c4
34b8 : 8d 07 d4 ac ee 07 c8 8c  f7
34c0 : 0b d4 68 8d 03 d0 68 8d  75
34c8 : 02 d0 a9 0b 8d 15 d0 ba  38
34d0 : e4 02 f0 1c 68 8d 01 d4  16
34d8 : 68 8d 00 d4 ac ee 07 c8  91
34e0 : 8c 04 d4 68 8d 05 d0 68  c6
34e8 : 8d 04 d0 a9 0f 8d 15 d0  34
34f0 : 4c 00 33 a2 a6 86 fc a2  35
34f8 : 30 86 fd ad ee 07 29 04  74
3500 : f0 09 a9 10 8d ee 07 85  59
3508 : 02 10 2f ad ee 07 29 70  40
3510 : d0 0d a9 d8 85 fc a9 14  fb
3518 : 8d ee 07 a0 0a 10 2b ad  1b
3520 : ee 07 0a 8d ee 07 85 02  07
3528 : c9 40 f0 04 a9 00 f0 02  30
3530 : a9 0a a2 27 9d 58 da ca  09
3538 : 10 fa a5 fc 18 69 0a 85  ce
3540 : fc a5 02 0a 85 02 90 f2  61
3548 : a0 0a b1 fc 99 97 06 88  79
3550 : d0 f8 4c bd 32 a2 0d bd  4f
3558 : e8 06 29 7f 9d e8 06 ca  4c
3560 : 10 f5 ae ef 07 e0 0f d0  6a
3568 : 03 20 ab 35 e0 7f d0 05  64
3570 : 20 ab 35 a2 ff 8a 18 69  8f
3578 : 10 8d ef 07 8d 18 d4 0a  2d
3580 : 85 02 a2 06 86 fd a2 e8  24
3588 : 86 fc a2 03 a5 02 0a 85  33
3590 : 02 90 0b a0 03 b1 fc 09  75
3598 : 80 91 fc 88 10 f7 a5 fc  82
35a0 : 18 69 05 85 fc ca d0 e4  92
35a8 : 4c bd 32 ad f2 07 49 07  b0
35b0 : 8d f2 07 8d 17 d4 a0 6e  a1
35b8 : b9 f8 da 49 0e 99 f8 da  14
35c0 : 88 10 f5 60 4d 41 52 54  ab
35c8 : 49 4e 20 41 48 4c 42 4f  f7
35d0 : 52 4e 20 33 34 31 38 20  a6
35d8 : 55 53 4c 41 52 fa a2 a6  e7
```

Listing 2. »SYNTHESIZER – OBJ« ist die Maschinensprache-Routine.

BASIC-KURS

Basic-Freunde aufgepaßt: Hier ist ein Super-Kurs, der Ihnen den Einstieg in die faszinierende Welt der Computer-Programmierung leicht macht. Praktische Beispiele begleiten Sie Schritt für Schritt.

Wer sich einen C64 gekauft hat, kann auf eine riesige Auswahl an Programmen zurückgreifen. Für fast alle denkbaren Anwendungen gibt es fertige Programme: Nach dem Motto »Diskette rein, und los geht's« erfährt man sehr schnell, welche Fähigkeiten in diesem Computer stecken.

Oft stellt man aber bei der Arbeit beispielsweise mit »seiner« Textverarbeitung fest, daß eine Funktion fehlt, die man gerade in diesem Moment benötigt. Oder das Programm zur Verwaltung Ihrer Disketten ist zwar gut, aber Sie wollen weitere Sortiermöglichkeiten. Leicht lassen sich viele Beispiele finden, in denen ein fertiges Programm zwar hervorragende Leistungen bietet, individuellen Wünschen aber nicht genügt.

Spätestens in diesem Moment wächst der Wunsch, eine Anwendung selbst zu programmieren. Diese kann den eigenen Wünschen optimal angepaßt werden. Doch viele Anfänger scheuen den Schritt zur Programmierung, weil sie glauben, dies sei für sie zu kompliziert. Leider bestärken manche Einführungsbücher dieses Vorurteil, wenn diese zu viele Kenntnisse voraussetzen und eher vom Lernen abschrecken. Das ist bedauerlich, denn die Programmiersprache Basic ist gerade für den Anfänger ideal. Genau dafür ist diese Sprache nämlich entwickelt worden: Basic bedeutet »Beginners All-purpose Symbolic Instruction Code«. Frei übersetzt heißt das »Einsteiger-Programmiersprache für alle Zwecke«.

Unser Kurs will beweisen, daß dies stimmt. Er wird Ihnen »Schritt für Schritt« helfen, sich auf diesem für Sie neuen und spannenden Gebiet zurechtzufinden. Dabei ist alles so geschrieben, daß Sie nur ein Minimum an Voraussetzun-



gen mitbringen müssen. Es wäre gut, wenn Sie die ersten Seiten Ihres C64-Handbuches schon durchgesehen hätten, da wir uns hier nicht mit dem Anschluß und der Inbetriebnahme der einzelnen Geräte beschäftigen können. Im übrigen haben wir versucht, unsere Erläuterungen möglichst allgemeinverständlich zu halten. Dies bedeutet, daß Fachausdrücke auf das unbedingt erforderliche Mindestmaß beschränkt bleiben sollen und in jedem Fall erklärt werden.

Der Kurs beginnt ganz langsam, mit vielen einfachen Beispielen und Wiederholungen. Damit das auf die Dauer aber nicht zu langweilig wird für Sie, nimmt das Tempo bei späteren Lektionen allmählich zu. Wie schnell Sie vorgehen, bestimmen letztlich Sie.

Seien Sie aber nicht zu ungeduldig und überfordern Sie sich nicht. Üben Sie ruhig den Stoff der einzelnen Lektionen ein wenig, bevor Sie weitergehen; so finden Sie am besten die für Sie geeignete Lerngeschwindigkeit heraus. Auf keinen Fall sollten Sie vergessen, gelegentlich eine Pause einzulegen; häufige kleinere Lernportionen garantieren einen besseren Erfolg, als Marathonsitzungen.

Der Kurs ist so aufgebaut, daß Sie bereits unmittelbar beim Durchlesen der Lektionen einzelne Aufgaben in den Computer eintippen und ausprobieren können. Die jeweiligen Stellen sind im Text mit dem Zeichen » → « gekennzeichnet. Am besten setzen Sie sich also gleich mit diesem Heft vor den eingeschalteten Computer. Gerade für Anfänger ist die positive Eigenschaft von Basic, kleinste Programmteile sofort ausprobieren zu können, von großem Nutzen und soll hier reichlich Anwendung finden. Damit es dabei dennoch geordnet zugeht, werden wir alles schön der Reihe nach besprechen und Ihnen auch zeigen, wie man ein Programm systematisch entwickelt, eben »Schritt für Schritt«.

Insgesamt besteht unser Kurs aus 34 Lektionen (siehe Seite 50/51). Sie werden darin die wichtigsten Befehle des Commodore-Basic 2.0 kennenlernen und üben (eine praktische Übersicht ist auf Seite 51). Es beginnt mit Erläuterungen der Tastatur und einfachen Eingabe- und Rechenübungen. Nach Darstellung des wichtigen PRINT-Befehles und einiger Vorbereitungen werden wir ein erstes Programm schreiben – ein kleines Spielchen mit Zahlen. Wenn wir unsere Programmierkenntnisse durch weitere Befehle und Funktionen ausgebaut haben, machen wir uns an eine etwas schwierigere Aufgabe: die Programmierung eines Ratespiels für Begriffe; später folgt eine Simulation des bekannten Spieles »Stadt - Land - Fluß«. Nachdem wir uns mit den Grundregeln der Dateiverwaltung, dem Speicheraufbau des C64 sowie mit der Technik von Unterprogrammen vertraut gemacht haben, steht uns schließlich noch die Erstellung einer vollständigen Dateiverwaltung bevor. Als Beispiel haben wir eine Literaturdatei gewählt. Danach wird es Ihnen sicher leichtfallen, ähnliche Programme für die Verwaltung Ihrer Adressen, Schallplatten oder Computer-Programme selbst zu schreiben.

Gewissermaßen als Zugabe folgen dann drei Lektionen zu verschiedenen Arten von Funktionen aus Arithmetik, Geometrie und formaler Logik.

Den Abschluß bildet eine Lektion über Fehlersuche, denn trotz aller Mühe werden Sie bald herausfinden, daß nicht immer alles auf Anhieb klappen wird. Ein Computer macht eben stets genau das, was man ihm sagt – und das ist leider nicht immer dasselbe, was man eigentlich will. Damit Sie also im Falle eines Falles den »Wurm« in einem fehlerhaften Programm leichter finden können (im Computer-Englisch spricht man übrigens von einem »bug«, d.h. Wanze oder Käfer),





verraten wir Ihnen in dieser Lektion wichtige Tricks. Nun aber genug der vielen Vorreden. Ich hoffe, Sie haben einen kleinen Vorgeschmack bekommen auf das, was Sie in unserem Kurs erwartet. Das soll fürs erste genügen, denn jetzt soll's endlich losgehen.

1 Wie bediene ich den Computer?

Voraussetzung für die Arbeit mit einem Computer ist eine Einrichtung, die dem Computer mitteilt, was er tun soll, und eine andere Einrichtung, um zu sehen, was er tut beziehungsweise was er getan hat. Dazu dient als allererstes die Tastatur und der Bildschirm. Weitere Eingabegeräte sind zum Beispiel der Joystick und die »Maus«, ein anderes wichtiges Ausgabegerät ist der Drucker. Wir wollen uns hier jedoch auf Tastatur und Bildschirm beschränken. Beide sind, sobald Sie den Computer einschalten, direkt miteinander verbunden.

Im Handbuch Ihres C 64 haben Sie sicher schon nachgelesen, wie die Tastatur funktioniert und wie damit Zeichen auf den Bildschirm gezaubert werden. Ich will das nicht im Detail wiederholen, aber ein paar Gedanken ist das Thema »Tastatur« schon wert.

Ich teile die Tasten in drei verschiedene Typen ein:
- Funktionstasten
- Zeichentasten
- Steuertasten

Mit *Funktionstasten* werden die vier senkrecht untereinander angeordneten Tasten <F1> bis <F7> rechts außen bezeichnet. Sie sind universell für alle möglichen Funktionen einsetzbar, können diese Funktionen aber erst dann erfüllen, wenn sie entsprechend programmiert worden sind. Aus diesem Grund werde ich sie später behandeln, und ich lasse sie vorläufig links, genauer gesagt: rechts liegen.

Zeichentasten erzeugen, wie ihr Name sagt, Zeichen. Dazu gehören Buchstaben und Zahlen, mathematische Zeichen (Addition, Multiplikation, runde Klammern etc.), Symbole (Dollar, Anführungsstrich, Pfeile und so weiter) und grafische Zeichen. Die grafischen Zeichen stehen übrigens bei den älteren C 64-Geräten und beim C 128 nicht oben auf den Tasten, sondern auf der Vorderseite.

Jede Zeichentaste bietet mehrere Zeichen und Symbole zur Auswahl. Um die verschiedenen Zeichen einer Taste auszuwählen und auf den Bildschirm zu bringen, verwenden wir spezielle Tasten wie <SHIFT>, <CTRL> und so weiter. Diese Tasten nenne ich *Steuertasten*.

Sie steuern auf sehr direkte Art und Weise alle Vorgänge auf dem Bildschirm. Aber sie erlauben auch, innerhalb von Programmen deren Abläufe zu steuern und zu verändern.

Ich halte diese Tasten für wichtig genug, um trotz ihrer Behandlung im Handbuch von Commodore etwas näher auf sie einzugehen.

Steuertasten sind also die CTRL-Taste, mit der die acht verschiedenen Zeichenfarben schwarz bis gelb eingestellt werden können, aber auch die SHIFT-Taste, welche den rechten Teil der Zeichen und grafischen Symbole umschaltet. Dazu gehört auch die Taste links unten, die das Firmenzeichen von Commodore trägt und welche die linken Symbole umschaltet.

Dazu gehören schließlich und letztlich die Cursor-Steuertasten <CRSR>, die INST/DEL-Taste und die CLR/HOME-Taste.

Diese letztgenannten Tasten sind eng mit der Wirkungsweise des sogenannten Editors verbunden.

Editor ist ein englisches Wort und heißt soviel wie Redakteur. In unserem Fall ist der Editor ein im Computer fest eingebautes Programm, welches wie ein Redakteur dafür sorgt, daß auf dem Bildschirm alle Zeichen und Symbole in den entsprechenden Farben auf den richtigen Platz kommen.

Ein sichtbares Hilfsmittel, dem Editor unsere Wünsche und Vorstellungen mitzuteilen, ist der *Cursor*.

Die Bedeutung und seine Handhabung haben Sie ja sicher schon dem Handbuch entnommen. Das Wort »Cursor« kommt nicht vom englischen »curse« = verfluchen (obwohl das Verhalten des Cursors oft dazu verleitet), sondern aus dem Lateinischen, wo es »Läufer« bedeutet.

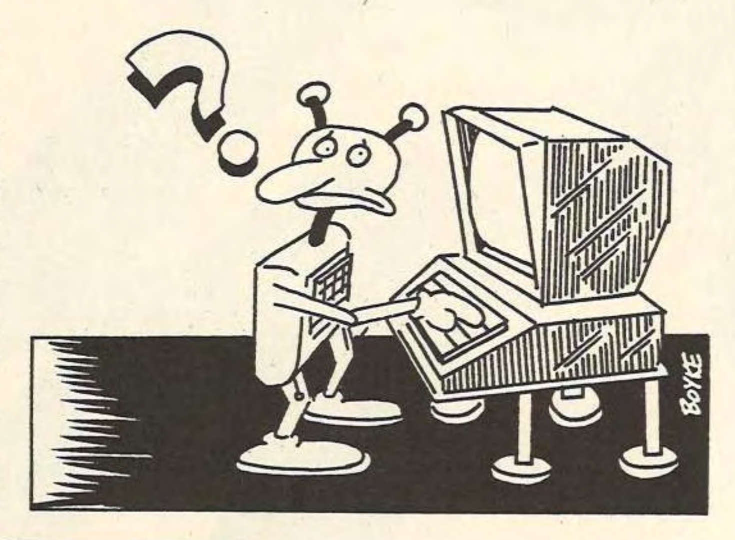
Der Cursor läuft also über den Bildschirm, gesteuert vom Editor. Sie können seinen Lauf auch steuern, und zwar mit den beiden Cursor-Steuertasten.

Eine andere Steuerung des Cursors bietet die CLR/HOME-Taste rechts oben. Allein gedrückt – dann gilt <HOME> – bringt sie den Cursor »nach Hause« nämlich in die linke obere Ecke des Bildschirms. Wird die Taste mit <SHIFT> umgeschaltet, bedeutet sie <CLR> – das heißt <CLEAR>, auf deutsch Freimachen oder Löschen. Jetzt wird nämlich der Cursor nach links oben gebracht und zusätzlich der Bildschirm gelöscht.

Probieren Sie bitte diese Cursor-Bewegungen aus.

→ Am besten fahren Sie den Cursor in die Mitte des Bildschirms, tippen ein paar beliebige Buchstaben ein und drücken dann die HOME-Taste.

BASIC-KURS:



ÜBERSICHT

ÜBERSICHT DER BASIC-BEFEHLE

→ Wiederholen Sie das Ganze und verwenden Sie dann die CLR-Taste.

Der Editor bietet uns noch einen anderen sehr lobenswerten Service. Er berücksichtigt nämlich, daß wir alle sehr menschliche Wesen sind, die nicht nur viele Fehler machen, sondern auch immer wieder ihre Meinung ändern. Der Editor erlaubt uns, Fehler zu korrigieren oder bereits getippte Zeichenfolgen abzuändern.

→ Sie können mit dem Editor ungestraft über vorhandene Zeichen fahren, ihn nach Lust und Laune anhalten und dort, wo er gerade blinkt, ein neues Zeichen eintippen.

Ich nenne das »überschreiben«. Wo wir ein Überschreiben nicht anwenden können, hilft uns die INST/DEL-Taste (rechts oben) weiter.

<INST> ist die Abkürzung für Insert, das heißt soviel wie einfügen.

<DEL> bedeutet Delete, und das heißt entfernen oder auslöschen.

Im Handbuch steht nur eine kurze Beschreibung der Wirkung dieser Taste, die aber nicht unbedingt ausreichend ist. Schon erste Versuche zeigen, daß die Taste ihre Tücken hat. Ich bin dafür, DEL und INST einfach auszuprobieren.

→ Schalten Sie den Computer aus und dann wieder ein.

→ Fahren Sie mit dem Cursor auf das A von READY. Er steht jetzt an der dritten Stelle vom linken Bildrand.

→ Drücken Sie die DEL-Taste. Das E ist verschwunden, es steht nur noch RADY da, der Cursor blinkt immer noch auf dem A, er ist aber jetzt an die zweite Stelle vom linken Bildrand gerückt.

→ Ein zweiter Druck auf die DEL-Taste reduziert das Wort auf ADY, und der Cursor steht jetzt auf der ersten Stelle.

Die DEL-Taste löscht also das links neben dem Cursor stehende Zeichen und verschiebt den Cursor mitsamt dem ganzen rechten Schwanz eine Stelle nach links. Was passiert aber, wenn der Cursor am linken Bildrand angestoßen ist?

→ Ein dritter Druck auf die DEL-Taste bringt den Cursor an den rechten Bildrand eine Zeile darüber, nur da gibt es gerade nichts zu löschen. Seinen ADY-Anhang läßt er am Anfang der unteren Zeile zurück.

→ Lassen Sie die DEL-Taste gedrückt. Der Cursor läuft kontinuierlich weiter nach links, löscht alles, was ihm in den Weg kommt - solange, bis er »zu Hause« links oben angekommen ist.

Der geSHIFTete Teil dieser Taste, nämlich <INST>, ist ebenso trickreich.

→ Schalten Sie den Computer aus und wieder ein.

→ Fahren Sie mit dem Cursor wieder auf das A von READY.

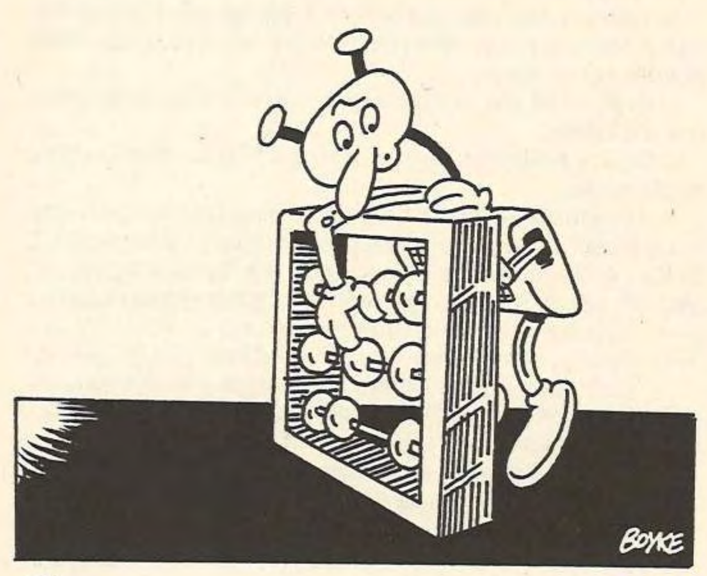
→ Drücken Sie die INST-Taste (SHIFT + INST/DEL). Der Cursor bleibt auf seiner dritten Stelle vom linken Bildrand stehen. Aber das Zeichen, auf dem er blinkt und alles, was rechts von ihm steht, wird eine Stelle nach rechts geschoben. Wo der Cursor blinkt, entsteht eine freie Stelle, auf die direkt ein neues Zeichen geschrieben werden kann.

→ Durch mehrfaches Drücken der INST-Taste werden mehrere Stellen freigeschoben, in die mehrere Zeichen hintereinander eingefügt werden können.

Dieses Freimachen beziehungsweise Einfügen geht nicht in beliebiger Länge, da der Editor gewissen Einschränkungen unterliegt. Ich würde natürlich am liebsten jetzt weiter über den Editor mit allen seinen Regeln und Einschränkungen dozieren. Und irgendwann muß ich es auch noch tun, denn beim Experimentieren werden Sie sicher noch darauf stoßen und wissen dann vielleicht nicht weiter.

Damit Sie aber nicht ungeduldig werden, möchte ich auf weitere Feinheiten vorläufig verzichten und das bisher Gelernte für die ersten Schritte in Basic anwenden.

Merken wir uns:
Die Zeichen- und Steuertasten erzeugen eine direkte Wirkung auf dem Bildschirm. Der Bildschirm wird von einem eingebauten Programm verwaltet - dem Editor. Er ermöglicht beliebige Änderungen und Korrekturen von bereits eingegebenen Texten und Zahlen.



2 Computer bedeutet Rechenmaschine

Diese Überschrift weist uns den Weg für das weitere Vorgehen.

Wenn jeder simple Taschencomputer rechnen kann, dann müssen es die Commodore-Computer auch können. Dazu will ich Ihnen noch schnell die Symbole der mathematischen Funktionen aufschreiben, die der Computer verwendet.

FUNKTION	SYMBOL	BEISPIEL
Addition	+	3+2
Subtraktion	-	3-2
Multiplikation	*	3*2
Division	/	3/2
Potenz	↑	3↑2

Die unterste Funktion ist wahrscheinlich die für Sie ungewohnteste. Das Beispiel wird im Klartext als »drei hoch zwei« oder als »drei Quadrat« gelesen.

Beim Taschenrechner wird normalerweise eine Rechnung so eingegeben:

→ 3 + 2 =

Aber beim C64 rührt sich da gar nichts, außer daß diese Zeile auf dem Bildschirm steht.

Und warum passiert nichts?

Überlegen Sie - alles, was wir gemacht haben - war, per Tastendruck Zahlen und Symbole einzutippen. Wir haben lediglich den Editor auf Trab gehalten, den Computer selbst haben wir damit nicht aufgeweckt.

Dieser Faulpelz schläft nämlich so lange, bis er einen Auftrag bekommt, etwas auszuführen. Für diesen Tritt, der ihn hinter dem Ofen hervorscheucht, brauchen wir eine der vorher zwar erwähnten, aber noch nicht eingesetzten Steuertasten zur Programmsteuerung.

Sie sitzt am rechten Rand der Tastatur und ist mit <RETURN> gekennzeichnet.

Schreiben wir also die Zeile von oben noch einmal. Dieses Mal bitte ich Sie aber, nicht die Eingabe-Technik der Taschencomputer zu verwenden, sondern mir eine »lesbare« Schreibweise nachzumachen - Sie werden gleich sehen, warum.

Schließen Sie die Eingabe mit der RETURN-Taste ab. Das sieht dann so aus:

→ SUMME = 3 + 2
→ Drücken Sie die RETURN-Taste

Aber wir sehen ja noch immer nichts!

Aller Anfang ist schwer. Wir müssen berücksichtigen, daß der Computer nichts, aber auch gar nichts von allein macht.

Für alles braucht er eine Anweisung. Nun, oben haben wir ihm die Anweisung gegeben, die Summe aus 3 plus 2 zu bilden. Das hat er auch gemacht, glauben Sie es mir. Aber wir haben ihm nicht gesagt, was er mit der Summe machen soll, und so hat er sie sich einfach nur gemerkt.

Basic-Befehl Nr. 1 PRINT
druckt alles, was hinter dem Wort »PRINT« steht, auf dem Bildschirm aus.

Wir wollen sie natürlich auf dem Bildschirm sehen. Dazu müssen wir dem Computer eine spezielle Anweisung geben.

Dieser Befehl, irgend etwas auf dem Bildschirm auszudrucken, lautet in Basic

PRINT.

Nach PRINT, was auf deutsch »drucken« heißt, geben wir dem Computer an, was er ausdrucken soll - in unserem Beispiel die Summe.

→ PRINT SUMME
→ Drücken Sie die RETURN-Taste.

Heureka! Endlich sind wir weitergekommen. Auf dem Bildschirm steht, vom Editor in die nächste Zeile gebracht, die Zahl, welche der Computer für die SUMME ausgerechnet hat.

Ich schlage vor, daß Sie mit dem ersten Wort der Sprache Basic, das Sie nun gelernt haben, noch ein bißchen üben.

Bilden Sie das Produkt von 15 x 14. Das geht doch einfach, oder?

→ PRODUKT = 15 * 14
→ nicht vergessen <RETURN> zu drücken
(Sie wissen doch, mit RETURN geben wir dem Computer die Anweisung, die getippte Zeile auszuführen)
→ PRINT PRODUKT
→ <RETURN> drücken

Wenn Sie gut Kopfrechnen können, dürfen Sie das Resultat nachprüfen. Oder vielleicht nehmen Sie einen Taschenrechner!

Ich wette, irgendein Leser sagt jetzt: »Was soll der Quatsch. Man kann doch die Rechnung 15*14 direkt mit PRINT ausführen. Der Umweg über SUMME = oder PRODUKT = ist doch völlig unnötig«. Tja, liebe Leser, das stimmt in der Tat. Diese Kurzform ist erlaubt, und ich will Sie Ihnen gleich vorführen, vielleicht mit der Division. Wir wollen 3 durch 2 teilen. Geben Sie dazu ein:

→ PRINT 3/2
→ <RETURN> drücken

Wir erhalten sofort den Wert 1,5 - aber halt!! Er ist 1.5, also nicht mit Komma, sondern mit Dezimalpunkt geschrieben.

Das ist die amerikanische Schreibweise, an die Sie sich gewöhnen müssen; als Preis dafür, daß Sie mit einem amerikanischen Computer arbeiten.

Also – der Befehl PRINT führt eine nachfolgende Rechnung direkt aus und druckt das Resultat auf den Bildschirm. Ist demnach meine oben genannte Methode, die Rechnung in zwei Schritten auszuführen, tatsächlich Quatsch?

Meine Antwort ist ein klares Nein. Beide Methoden haben ihre Berechtigung, und ich will Ihnen auch den Unterschied zeigen.

Mit der ersten Methode (wir haben sie bei SUMME und PRODUKT angewendet) erhält der Computer zuerst den Auftrag, eine Rechnung durchzuführen. Das Resultat merkt er sich unter dem Stichwort SUMME beziehungsweise PRODUKT. Haben Sie es gelesen? Er merkt sich das Resultat und legt es in einem Speicher ab. Das bedeutet, daß wir es mit PRINT so oft wir wollen, auf den Bildschirm drucken können. Geben Sie jetzt gleich noch mal ein:

→ PRINT SUMME
→ <RETURN> drücken

Und siehe da, das Resultat von vorhin erscheint wieder. Dasselbe geht mit dem Stichwort PRODUKT.

Sie brauchen übrigens die Zeile nicht neu einzugeben. Vergessen Sie den Service des Editors nicht!

→ Fahren Sie mit dem Cursor auf die letzte PRINT-Zeile und überschreiben Sie das Wort SUMME mit dem Wort PRODUKT, dann muß nur noch die RETURN-Taste gedrückt werden.

Und noch einmal siehe da, wir erhalten das Resultat der Multiplikation wieder.

So, und was ist mit der Division?

Leider, leider! Da geht nichts mehr. Wir haben kein Stichwort für das Resultat vorgegeben, und der Computer hat sich auch nichts gemerkt.

Das ist also der Unterschied:

• PRINT 3 + 2	• SUMME = 3 + 2 • PRINT SUMME
Das Resultat der Rechnung wird ausgedruckt	Das Resultat der Rechnung wird unter dem Stichwort SUMME gespeichert und dann ausgedruckt

Die linke Anwendung des Befehls PRINT ist einfach, und Sie können sie immer verwenden, um nach Art des Taschenrechners schnell einmal irgendwelche Rechnungen zu machen. Aufgezeichnet wird das Resultat allerdings nur auf dem Bildschirm. Im Computer selbst bleibt nichts gespeichert.

Die rechte Art ist schwieriger, aber interessanter. Es ist eigentlich erstaunlich, daß wir dem Computer die Anweisung geben, sich eine Zahl unter einem Namen oder – wie ich es vorhin genannt habe – unter einem Stichwort zu merken, ohne einen Basic-Befehl geben zu müssen.

Es gibt tatsächlich einen Befehl dafür, er heißt:

LET

Die Anweisung mit dem Stichwort »SUMME« sieht unter Zuhilfenahme des Befehls LET so aus:

```
LET SUMME = 3 + 2
```

In unseren Sprachgebrauch übersetzt heißt das ungefähr: »Laß die Summe gleich dem Resultat von 3 plus 2 sein«. In der Computersprache nennt man das eine ZUWEISUNG.

Wir können demnach beliebige Stichwörter hernehmen und ihnen mit dem LET-Befehl Zahlenwerte zuweisen, der Computer merkt sie sich alle.

Wie würden Sie zum Beispiel dem Stichwort A (kürzer kann es nicht sein) den Wert 375 zuweisen, dann dem Stichwort X den Wert 15 und schließlich die Division von vorhin, die der Computer sich nicht gemerkt hat, jetzt dauerhaft durchführen?

Versuchen Sie es:

So muß es aussehen:

→ Löschen Sie den Bildschirm (<SHIFT>+ <CLR/HOME>)
→ LET A = 375 (<RETURN> drücken)
→ LET X = 15 (<RETURN> drücken)
→ LET QUOTIENT = 3/2 (<RETURN> drücken)

Jetzt sind im Computer die zwei Werte unter den Stichworten A, X und das Ergebnis der Division unter dem Namen QUOTIENT gespeichert.

Mit PRINT-Befehlen können wir alle drei auf den Bildschirm bringen:

→ PRINT A (<RETURN> drücken)
→ PRINT X (<RETURN> drücken)
→ PRINT QUOTIENT (<RETURN> drücken)

SUMME, PRODUKT, A, X und QUOTIENT, die wir bisher »Stichwörter« oder »Namen« genannt und welchen wir mit dem LET-Befehl einen Wert zugewiesen haben, werden mit einem eigenen Fachausdruck bezeichnet.

Sie heißen Variable. Ich werde später noch mehr über sie berichten. Zunächst bitte ich Sie lediglich, sich den Namen zu merken – manchmal werde ich vielleicht auch noch ein Stichwort sagen.

Wozu brauchen wir den Befehl »LET«? Am Anfang haben wir ohne ihn den Stichwörtern Resultate zugewiesen!

In Basic kann in der Tat der Befehl LET weggelassen werden. Wichtig ist lediglich, daß zuerst die Variable dasteht, gefolgt von dem Gleichheitszeichen (=) und dahinter der Wert (oder der durch die Rechnung auszurechnende Wert).

Basic-Befehl Nr. 2 LET
weist einer Variablen einen Wert zu. Er kann auch weggelassen werden. Es genügt der Name der Variablen, ein Gleichheitszeichen und der zugewiesene Wert.

Interessant ist übrigens, daß der Computer sich ein und dasselbe Stichwort nur einmal merkt. Wenn Sie zuerst dem Stichwort QUOTIENT den Wert 3/2 zuweisen und gleich danach den Wert 15/3, erhalten Sie mit dem PRINT-Befehl nicht die Zahl 1.5, sondern 5.

Das erste Resultat hat der Computer in seinem Speicher durch das zweite überschrieben. Probieren Sie es aus:

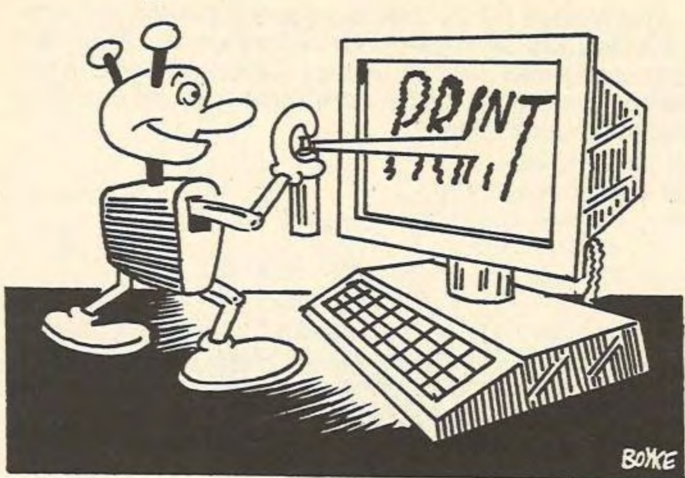
→ PRINT QUOTIENT (<RETURN> drücken)
→ LET QUOTIENT = 15/3 (<RETURN> drücken)
→ PRINT QUOTIENT (<RETURN> drücken)

Zuerst kommt der alte Quotient von vorhin, danach der neue Wert.

Merken wir uns:

1. Nicht nur wir können Zeichen auf den Bildschirm bringen – der Computer selbst kann das auch. Der Befehl PRINT druckt alles, was dahintersteht, auch Ergebnisse von Rechenanweisungen, auf den Bildschirm.
2. Unter einem Stichwort (Variable genannt) kann sich der Computer Zahlenwerte merken, die beliebig oft mit dem PRINT-Befehl ausgedruckt werden können. Voraussetzung ist lediglich, daß mit dem LET-Befehl dem Stichwort der Zahlenwert zugewiesen wird. Das Wort »LET« selbst kann auch weggelassen werden.





3 PRINT – der Universalbefehl

3.1 Zahlenwerte ausdrucken

Bisher haben wir nur Zahlenwerte und zugehörige Variable mit PRINT auf dem Bildschirm ausgedruckt.

Ist Ihnen das viele PRINT-Tippen, so wie bei den Beispielen oben, auch so lästig wie mir?

Diesen Dauerversuch mit unzähligen PRINT-Befehlen können Sie vereinfachen. Alle Commodore-Computer sind nämlich Meister der Abkürzung. So drastisch, wie wir den Befehl LET durch Weglassen abgekürzt haben, dürfen wir den Befehl PRINT nicht behandeln. Aber immerhin: Er wird mit dem Fragezeichen abgekürzt.

Das Befehlswort »PRINT« kann durch das »?« ersetzt werden.

Statt PRINT 123
dürfen Sie schreiben ? 123

Machen Sie bitte Gebrauch von dieser Abkürzung, wann immer Sie wollen.

Ich werde allerdings stets das ganze Befehlswort verwenden, damit der Text gut lesbar bleibt.

Die Sprache Basic gewährt uns noch eine Marscherleichterung. Es ist nämlich erlaubt, mehrere auszudruckende Werte hinter einen einzigen PRINT-Befehl zu setzen.

Zuerst will ich Ihnen zeigen, wie Sie es nicht machen dürfen – probieren geht über studieren.

Haben Sie die alten Stichwörter noch im Computer? Wenn ja, dann brauchen Sie die folgenden Zuweisungen nicht noch mal eingeben. Ich wiederhole sie für alle Leser, die inzwischen ihren Computer ausgeschaltet und damit sein Gedächtnis gelöscht haben.

→ A = 375 (<RETURN> drücken)
→ X = 15 (<RETURN> drücken)
→ QUOTIENT = 15/3 (<RETURN> drücken)

Sie sehen, ich habe LET weggelassen.

So, die Werte haben ihr Stichwort, unter dem sie der Computer in seinem Speicher wiederfindet. Jetzt wollen wir sie mit einem einzigen PRINT-Befehl ausdrucken:

→ PRINT A X QUOTIENT (<RETURN> drücken)

Der Computer druckt eine Null aus. Wie gesagt, so geht's nicht.

Wenn ich Ihnen sage, warum eine Null erscheint, werden Sie gleich auf die Lösung des Problems kommen. Was passiert da?

Für den Computer ist eine Leerstelle (zwischen den Variablen) genauso ein Zeichen wie jeder Buchstabe. Das heißt, daß er nach dem Drücken der RETURN-Taste in seinem Speicher nach einem Stichwort »A X QUOTIENT« sucht – und das findet er natürlich nicht.

Ihnen ist sicher klar, daß wir also die drei Stichwörter voneinander trennen müssen. Die Rechtschreibregeln von Basic erlauben keine Wörtertrennung mit Leerzeichen, sondern mit dem Semikolon »;«.

Verbessern Sie bitte (durch einfaches Einfügen von zwei Semikolons) die letzte Zeile oben:

→ PRINT A;X;QUOTIENT (<RETURN> drücken)

Jetzt stimmt's.

Bei diesem Beispiel kann ich Ihnen noch etwas zeigen, diesmal eine Eigenheit – oder besser gesagt – eine Vorsorge des guten alten Editors. Schauen Sie sich genau an, wie der Editor die drei Zahlen 375, 15 und 5 auf den Bildschirm geschrieben hat. Vor der ersten Zahl steht eine Leerstelle, zwischen den Zahlen sind zwei Leerstellen.

Das kommt daher, daß der Editor vor jeder Zahl eine Stelle reserviert, um im Fall einer negativen Zahl Platz für das Minuszeichen zu haben. Und zwischen den Zahlen hält er natürlich einen Abstand frei. Wie bereits gewohnt, wollen wir das am Computer überprüfen.

→ Bitte geben Sie die drei Werte unserer Variablen als negative Zahlen, also mit einem Minuszeichen ein. Wenn Sie die Zuweisungen noch auf dem Bildschirm stehen haben, geht es schneller mit Hilfe des Editors. Fahren Sie mit dem Cursor auf die jeweils vorderste Ziffer, schaffen Sie mit der INSERT-Taste (<SHIFT> + <INST/DEL>) eine Leerstelle und tippen Sie ein Minuszeichen ein, danach wie immer ein <RETURN>.
Fahren Sie mit dem Cursor auf die Zeile mit dem PRINT-Befehl und drücken Sie die RETURN-Taste.

Ja, der Editor macht es uns wirklich sehr bequem. Und jetzt sehen Sie auch, daß die reservierten Plätze im Ergebnis von den Minuszeichen belegt worden sind. Was bleibt, sind lediglich die Abstände – der guten Lesbarkeit zuliebe.

Das Semikolon ist nicht das einzige Symbol zur Trennung von Variablen hinter einem PRINT-Befehl. Die von Commodore verwendeten Versionen von Basic erlauben auch das Komma – allerdings hat es eine etwas andere Wirkung als das Semikolon.

→ Ersetzen Sie in der Zeile oben, welche den »dreifachen« PRINT-Befehl enthält, die zwei Semikolon durch Komma (direkt durch Überschreiben). <RETURN> nicht vergessen!

Wir sehen jetzt wieder die drei Zahlen, aber viel weiter auseinandergerückt.

Dabei wird der Bildschirm sozusagen in vier Teile geteilt. Die ausgedruckten Werte beginnen am linken Rand, dann ab dem 10., 20., und 30. Platz.

Merken wir uns:
Hinter einem einzigen PRINT-Befehl können mehrere Werte auf einmal ausgedruckt werden. Sie müssen entweder durch ein Semikolon oder ein Komma getrennt sein.
Das Semikolon bewirkt bei Zahlen einen »normalen« Abstand von zwei Leerstellen.
Das Komma verschiebt die Werte auf das nächste Bildschirm-Viertel.

3.2 Buchstaben und Zeichen ausdrucken

Was macht der PRINT-Befehl mit allen anderen Zeichen und Symbolen? Versuchen Sie es ruhig einmal. Fangen Sie mit den Tasten links oben an und arbeiten Sie sich durch.

Schon bei dem PRINT-Befehl der allerersten Taste, nämlich dem Symbol »Pfeil nach links«, reagiert der Computer unerwartet.

⟶ PRINT ← (<RETURN> drücken)
ergibt eine englische Meldung auf dem Bildschirm:

SYNTAX ERROR
READY

Mit der Meldung »SYNTAX ERROR« sagt uns der Computer, daß wir einen Rechtschreib- oder Satzbaufehler begangen haben. Das Zeichen »←« darf zum Beispiel nicht so einfach mit einem PRINT-Befehl verwendet werden.

Alle mathematischen Zeichen, Interpunktionen, die geSHIFTeten Symbole der Zifferntasten und andere reagieren in dieser Form.

Andere wieder, wie zum Beispiel mit SHIFT- oder Commodore-Taste (CBM-Taste) umgeschaltete Buchstaben (auf die auf der Vorderseite der Tasten angegebenen Grafiksymbole), ergeben gar keine Reaktion.

Den PRINT-Befehl mit Buchstaben haben wir ja schon mehrfach verwendet. In dem Beispiel ganz oben hat er allerdings nicht den Großbuchstaben A, sondern den Zahlenwert ausgedruckt, der unter dem Kennbuchstaben A im Computer gespeichert worden ist.

Versuchen Sie es mit einem anderen Buchstaben oder Wort, welchem wir noch keinen Wert zugewiesen haben, zum Beispiel:

⟶ PRINT WORT (<RETURN> drücken)

Wir erhalten eine Null.

Das heißt nichts anderes, daß der PRINT-Befehl schlicht und einfach jeden Buchstaben bzw. jede Buchstabenfolge als ein Stichwort - als eine Variable - interpretiert und deren Wert im Speicher sucht, um sie auszudrucken. Haben wir ihr noch keinen Wert zugewiesen, wie in unserem letzten Beispiel dem Wort »WORT«, dann druckt er eine Null aus.

Wenn der für den Bildschirm zuständige Editor (ein fest eingebautes Computerprogramm) bereitwillig alle Buchstaben druckt, die wir direkt mit der Tastatur eingeben, dann muß es der Computer mit dem PRINT-Befehl auch können.

Die Lösung für unser Problem könnte man als zungenbrechendes Kochrezept so hinschreiben:

Um dem Computer mitzuteilen, daß das Wort »WORT« ein Wort sein soll, muß man "WORT" schreiben. Das Geheimnis liegt im Gebrauch der Anführungszeichen!

⟶ PRINT "WORT" (<RETURN> drücken)

Für Zeichen, Zahlen, Buchstaben, kurz für alles, was zwischen zwei Gänsefüßchen steht, verwendet die Computersprache ein eigenes Wort: *Zeichenkette* oder auf englisch: *String*.

Strings sind demnach auch alle grafischen Symbole der Buchstabentasten, die auf der Vorderseite der Tasten zu sehen sind. Auch sie kann man per PRINT auf den Bildschirm bringen.

Für die linken Symbole müssen wir mit der Taste gleichzeitig die Commodore-Taste (unten ganz links), für die rechten Symbole die SHIFT-Taste drücken.

Die Schwierigkeit liegt nun darin, daß ich Ihnen im Text des Kurses diese speziellen Symbole nicht ohne weiteres angeben kann. Diese Commodore-spezifischen Zeichen und Symbole hat das Fotosatzsystem, mit dem dieses Heft erstellt wurde, verständlicherweise nicht zur Verfügung.

Daher werde ich an ihrer Stelle folgende Symbole anwenden:

- für die SHIFT-Taste <SHIFT>
- für die Commodore-Taste <CBM>
- (SHIFT W) bezeichnet das Symbol vorn rechts auf der W-Taste
- <CBMZ> bezeichnet das Symbol vorn links auf der Z-Taste

Ich zeige Ihnen die Verwendung dieser Symbole am besten an einem Beispiel:

⟶ PRINT "<SHIFT U><SHIFT I>"
(<RETURN> drücken)

dann druckt der Computer aus den beiden Viertelkreisen der U- und I-Taste einen Halbkreis aus.

Noch ein Beispiel:

⟶ PRINT "<CBM Q><CBM W>"
(<RETURN> drücken)

ergibt ein großes H.

Die Schreibweise zwischen Anführungszeichen erlaubt uns, nicht nur Buchstaben und deren umgeschaltete grafische Zeichen, sondern auch Ziffern und alle Symbole auszuPRINTen.

Verzeihen Sie mir bitte dieses saloppe Computerdeutsch. Im Grunde ist es ja nicht sehr schön, was wir da mit der deutschen Sprache machen, und ich müßte mich eigentlich schämen, daß ich derartige Wortschöpfungen verwende. Aber abgesehen davon, daß Computerdeutsch gewisse Vorgänge sehr kurz und präzise ausdrückt, verteidige ich mich gern damit, daß andere Gruppen unserer Gesellschaft ja auch ihre eigenen Ausdrücke haben, wie die Segler, Musiker und Jäger. Bei den letzteren meine ich natürlich nicht das Jägerlatein.

Wo waren wir stehengeblieben? Ach ja, beim PRINTen von Ziffern und Symbolen als Strings.

⟶ PRINT "()" (<RETURN> drücken)

Diese Anweisung druckt die beiden Klammern ohne SYNTAX ERROR aus. Alle Pfeile, Sterne, Interpunktionen und arithmetischen Symbole sind willig ausdruckbar, wenn sie in Anführungszeichen eingepackt sind.

Apropos Interpunktionen! Wir haben gelernt, daß mit dem Semikolon Stichwörter (Variable) voneinander getrennt werden können, selbst wenn sie hintereinander bei derselben PRINT-Anweisung stehen.

Zwischen Anführungszeichen verlieren sie diese Wirkung, sie sind dann nichts anderes als grafische Zeichen. Geben Sie ein:

⟶ A = 3 (<RETURN>)
⟶ B = 12 (<RETURN>)
⟶ C = 17 (<RETURN>)
⟶ PRINT A;B;C;"A;B;C" (<RETURN>)

Zuerst wird der Wert der drei Variablen A, B und C nebeneinander ausgedruckt, gefolgt von den fünf Symbolen zwischen den Anführungszeichen.

Mit dem Komma und seiner verschiebenden Wirkung ist es ebenso.

Und Ziffern? Ziffern gehen auch, aber sie gingen ja vorher schon, ohne Anführungszeichen.

Da ist aber ein ganz kleiner Unterschied. Passen Sie auf:

⟶ PRINT 123 (<RETURN>)
⟶ PRINT "123" (<RETURN>)

- Der erste PRINT-Befehl versteht die Ziffern 123 als Zahl und reserviert, wie früher schon besprochen, eine Stelle vorher für das Vorzeichen.
- Der PRINT-Befehl mit Anführungszeichen versteht die Ziffern 123 als String, der natürlich kein Vorzeichen braucht und daher auch keine Platzreservierung bekommt.

Dieser feine Unterschied ist recht wichtig, wenn Zahlentabellen oder Zahlengrafiken erzeugt werden sollen. Wir stoßen später sicher noch darauf.

Bevor ich ein weiteres Fazit ziehe, möchte ich Ihnen eine vorläufig letzte Anwendung des PRINT-Befehls zeigen.

Wir haben gesehen (und angewendet), daß eine auf den Bildschirm geschriebene Zeile erst dann zu einem Befehl für den Computer wird, wenn sie mit dem Drücken der RETURN-Taste »abgeschlossen« wird. Wir haben daher immer nur einen einzigen Befehl in eine Zeile geschrieben.

Die Frage stellt sich natürlich:

Können wir mehrere Befehle in
eine einzige Zeile schreiben??



Wie immer – probieren geht über studieren! Geben Sie ein:

→ PRINT "1.BEFEHL"   PRINT "2.BEFEHL"

→ drücken Sie die RETURN-Taste

Nun, SYNTAX ERROR sagt uns, daß diese Schreibweise nicht erlaubt ist.

Wie bei mehreren Variablen brauchen wir bei mehreren Befehlen auch ein Symbol zur Trennung. Bei Befehlen ist dies der Doppelpunkt. Fügen Sie bitte zwischen die beiden PRINT-Befehle oben einen Doppelpunkt ein und drücken Sie noch mal auf Return.

Es muß also da stehen:

→ PRINT "1.BEFEHL" : PRINT "2.BEFEHL"

Jetzt stehen die beiden »Befehle« ausgedruckt da – untereinander!

Durch den Doppelpunkt werden die zwei PRINT-Befehle so behandelt, als stünden sie in zwei getrennten Zeilen.

Und damit wird auch unsere anfängliche Rechnerei wesentlich einfacher – ich hoffe, Sie erinnern sich noch. Sonst schauen Sie bitte noch mal am Anfang nach.

Für die Berechnung der Summe 3 + 2 brauchen wir nur eine Zeile:

→ SUMME = 3 + 2 : PRINT SUMME
(<RETURN> drücken)

Damit soll's erstmal genug sein. Ich glaube, Sie haben jetzt eine ganze Reihe von Regeln gelernt, die für die ersten Schritte der Programmierung eines Commodore-Computers sehr wichtig sind. Vielleicht wollen Sie sich nun eine kleine Pause gönnen und zwischendurch Ihr Geschick an einem Computerspiel ausprobieren. Viel Spaß und bis gleich!

Merken wir uns:

1. Buchstaben werden vom PRINT-Befehl als Variable eingestuft, und entsprechend wird ihr Wert ausgedruckt.
2. Ziffern und Zahlen werden vom PRINT-Befehl als arithmetische Ausdrücke gewertet und erhalten dementsprechend ein Vorzeichen. Das positive Vorzeichen wird nicht ausgedruckt, eine Leerstelle ist dafür vorhanden.
3. Alle Symbole der Tastatur, welche zwischen Anführungszeichen stehen, werden vom PRINT-Befehl als grafische Zeichen erkannt und formgetreu ausgedruckt. Sie werden Zeichenketten beziehungsweise Strings genannt.
4. Um mehrere Befehle in eine einzige Zeile schreiben zu können, müssen sie durch einen Doppelpunkt voneinander getrennt sein.
5. Schreibfehler oder inkorrekte Anweisungen quittiert der Computer mit der Fehlermeldung SYNTAX ERROR.

4 Was ist ein Programm?

Bislang haben wir den Computer so benutzt, als wäre er ein Taschenrechner.

Wir haben alle Angaben, Befehle und so weiter direkt eingetippt; und der Computer hat sie nach dem Drücken der RETURN-Taste direkt ausgeführt.

Ich habe mit Absicht das Wort direkt gleich zweimal verwendet. Diese Art des Betriebs wird nämlich Direkt-Modus genannt.

Im Gegensatz dazu steht der sogenannte Programm-Modus.

Jetzt also taucht das Schlagwort Programm zum ersten Mal auf. Haben Sie schon sehnlich darauf gewartet? Es wäre kein Wunder, wollen wir doch Programmieren lernen.

In meinem alten Konversationslexikon steht unter Programm: Festfolge. Sie wissen, was damit gemeint ist, nämlich eine detaillierte Angabe der einzelnen Darbietungen und Aktivitäten einer Veranstaltung in ihrer genauen zeitlichen Reihenfolge. Denken Sie an das Programm in einem Zirkus.

Diese Definition läßt sich gut auf einen Computer übertragen.

Wenn ein Computer viele Befehle in einer bestimmten Reihenfolge nacheinander ausführen soll, müssen sie ihm als »Festfolge« vorgegeben sein. Diese Festfolge muß er auswendig kennen – was für ihn kein Problem bedeutet, besitzt er doch ein gutes Gedächtnis.

Dieses »Gedächtnis« haben wir ja schon kennengelernt. Die im Direkt-Modus eingegebenen Werte von Stichwörtern (Variablen) hat er in seinem Speicher festgehalten, und wir konnten sie jederzeit abfragen.

Ein Computer hat aber auch einen Programm-Speicher, in welchem er ein aus mehreren Befehlen bestehendes Programm festhält.

Alles, was wir tun müssen, ist, ihm die Befehle und ihre Reihenfolge einzugeben. Wie tun wir das?

Die Angabe der Reihenfolge ist sehr simpel. Wir geben einfach jeder Zeile, in der ein oder mehrere Befehle stehen, eine Nummer.

Die Eingabe der Befehle geht genauso wie vorher – mit der RETURN-Taste. Nur – vorher hat der Computer den Befehl sofort ausgeführt. Wenn aber eine Nummer davorsteht, dann führt er den Befehl nicht aus, sondern steckt ihn in seinen Programmspeicher.

Das machen wir jetzt gleich einmal. Geben Sie bitte unsere allerersten Rechenanweisungen mit Zeilennummern ein:

→ 1 SUMME = 3 + 2 (<RETURN>)

→ 2 PRINT SUMME   (<RETURN>)

Siehe da, er hat nichts ausgeführt, und ich behaupte, dieses kleine Programm steht jetzt im Programmspeicher.

Ich kann das auch beweisen. Es gibt einen Befehl in Basic, der alles, was im Programmspeicher steht, in der Reihenfolge der Zeilennummern auf dem Bildschirm ausdruckt. Er heißt

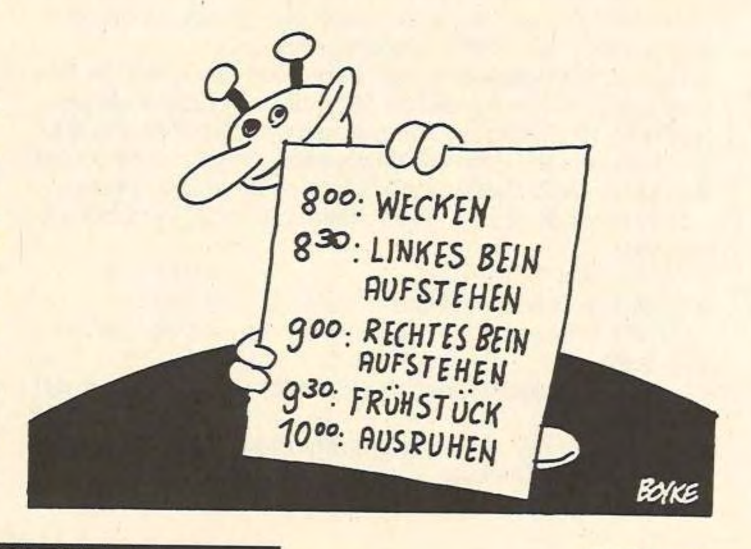
LIST

was auf deutsch soviel wie »auflisten« bedeutet.

→ Löschen Sie den Bildschirm mit der CLR-Taste

→ Tippen Sie direkt das Wort LIST ein und drücken Sie die RETURN-Taste.

Die beiden Anweisungen samt Nummer erscheinen auf dem Bildschirm. Was zu beweisen war.



Basic-Befehl Nr. 3 LIST
- druckt den Inhalt des Programmspeichers auf dem Bildschirm aus.
- Durch Angabe der Nummer oder eines Nummernbereiches können einzelne Befehle oder bestimmte Programmteile separat ausgedruckt werden.
- Durch den Befehl LIST wird das Programm nicht gelöscht.

Den ersten Satz dieser Befehlsbeschreibung haben wir ja schon praktiziert. Den zweiten Satz möchte ich vorläufig erst mal so stehenlassen. Zu seiner Erprobung brauchen wir längere Programme und die haben wir vorerst noch nicht.

So, unser Programm, bestehend aus zwei Befehlen, hat der Computer gespeichert. Jetzt soll er es ausführen!

Unsere alte Methode, den Computer hinter dem Ofen hervorzuscheuchen, war der Tritt mit der RETURN-Taste. Diese bringt uns hier aber überhaupt nichts. Was wir brauchen ist ein Befehl, der den Computer veranlaßt, das Programm auszuführen. Ein derartiger Befehl wird uns von Basic geboten. Er heißt

RUN

was mit »loslaufen« übersetzt werden könnte.

Basic-Befehl Nr. 4 RUN
- startet ein im Programmspeicher des Computers befindliches Programm.
- Mit RUN wird ein Programm nicht gelöscht. Der RUN-Befehl kann beliebig oft wiederholt werden.

→ Tippen Sie direkt das Wort RUN ein und drücken Sie die RETURN-Taste.

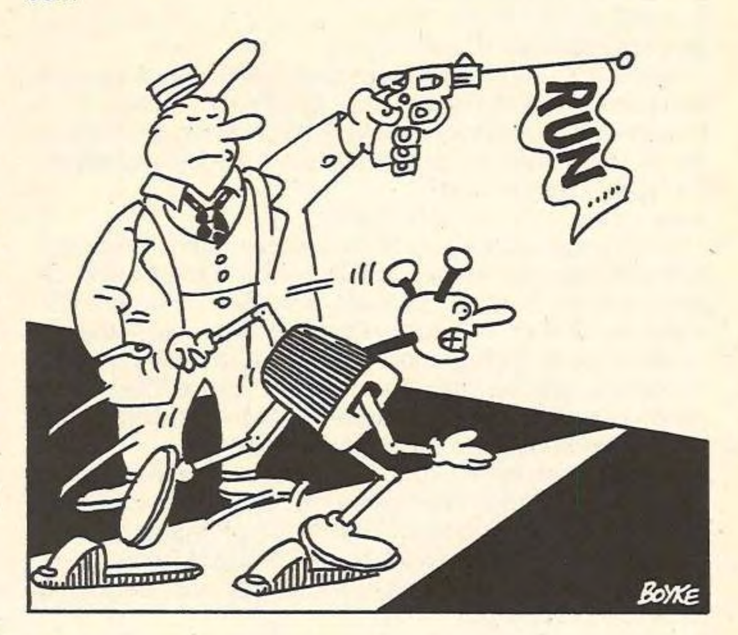
Und wie der Blitz, so schnell können Sie gar nicht schauen, steht das Resultat auf dem Bildschirm.

Na ja, sagen vielleicht einige Skeptiker unter Ihnen, im Direkt-Modus vorhin ging das auch nicht viel langsamer, besonders die Einzeiler-Version mit dem Doppelpunkt.

Richtig, aber wir haben ja nur zwei Befehle verwendet. Nehmen Sie mal 20 Befehle, da sieht die Sache schon anders aus. Abgesehen von der Geschwindigkeit gehen die überhaupt nicht in eine Zeile – und was dann?

Merken wir uns:
1. Der Computer hat einen Speicher mit mehreren getrennten Bereichen. Alle Variablen und ihre jeweiligen Werte stehen im Variablenspeicher. Programmzeilen stehen im Programmspeicher.
2. Immer wenn die RETURN-Taste gedrückt wird, überprüft der Computer die Zeile, in der sich gerade der Cursor befindet. Beginnt die Zeile mit einer Nummer, wird sie als Programmzeile erkannt und im Programmspeicher abgelegt.
   Hat sie keine Nummer am Anfang, dann führt sie der Computer im Direkt-Modus sofort aus.
3. Mit dem Basic-Befehl LIST werden alle im Programmspeicher stehenden Zeilen in Reihenfolge der aufsteigenden Zeilennummern auf dem Bildschirm ausgedruckt.
4. Ein Programm wird mit dem Basic-Befehl RUN gestartet.

Nun sollten wir uns schleunigst an ein längeres Programm wagen. Natürlich verwenden wir nur Dinge, die wir bisher schon verwendet beziehungsweise besprochen haben.



5 Ein erstes Programmbeispiel

Zuerst brauchen wir einen Plan, was programmiert oder besser gesagt, was vom Computer gemacht werden soll.

Ich schlage dazu eine Aufgabe mit folgenden Teilen vor:
1. Unter dem Stichwort A sollen zwei Zahlen multipliziert werden
2. Unter dem Stichwort B sollen zwei Zahlen dividiert werden
3. Das Stichwort C sei die Differenz der Werte von A und von B
4. Der Wert von A soll nicht als Zahl, sondern als Gleichung ausgedruckt werden.
5. Auch der Wert von B soll als Gleichung ausgedruckt werden.
6. Dann folgt eine Textangabe, wie der Wert von C berechnet wird. Dabei sollen im Text die jeweiligen Werte von A und B automatisch eingesetzt werden.
7. Als letztes wird auch der Wert von C als Gleichung ausgedruckt.

Das klingt schlimmer, als es ist. Wir müssen es einfach Schritt für Schritt angehen.

Für den Punkt 1 nehmen wir die Zahlen 24 und 3

→ `1 A = 24 * 3` (<RETURN>)

In dieser Zeile mit der Nummer 1 weisen wir dem Stichwort A das Resultat der Multiplikation von 24 x 3 zu. Wir hätten das auch mit dem Basic-Befehl LET machen können, aber wie gesagt, man kann ihn weglassen.

Punkt 2 verlangt etwas Ähnliches, wir nehmen die Zahlen 16 und 8.

→ `2 B = 16/8` (<RETURN>)

Auch Punkt 3 macht keine Ausnahme, nur verwenden wir jetzt statt absoluter Zahlen die Stichwörter A und B. Der Computer muß selbst im Lauf des Programms die Werte für A und B im Speicher heraussuchen und einsetzen.

→ `3 C = A - B` (<RETURN>)

Bis hierhin war alles so wie gehabt. Im Punkt 4 soll der Wert von A, also das Resultat aus Zeile 1, als »Gleichung« ausgedruckt werden.

Was ich damit meine, ist schnell gesagt. Der Befehl PRINT A würde nur den Zahlenwert von A auf den Bildschirm bringen. Ich möchte aber, daß das Programm diese fünf Zeichen hinschreibt:

A = 72

Wie erreichen wir das ?

Nun, die Zahl 72 ist der Wert von A, mit PRINT A zu erzielen. Davor steht ein Text! Damit der PRINT-Befehl »A =« hinschreibt, müssen wir das als String schreiben, das heißt wir müssen den Anführungszeichen-Modus anwenden. Das sieht dann so aus:

→ 4 PRINT "A =" A (<RETURN>)

Was zwischen den Anführungszeichen steht, druckt der Computer so aus, wie es ist. Das zweite A steht nackt da, also ist es ein Stichwort, dessen Wert der Computer direkt hinter den Text »A =« setzt - mit Vorzeichenstelle natürlich.

Wenn Sie es nicht glauben, oder es sich nicht vorstellen können, lassen Sie doch diesen ersten Teil des Programms gleich einmal laufen. Zur besseren Übersicht löschen Sie zuerst den Bildschirm mit der CLR-Taste.

Keine Angst, es wird nur der Bildschirm gelöscht, aber nicht das Programm im Programmspeicher. Um das nachzuprüfen, LISTen Sie es aus. Wenn das Programm schön dasteht, lassen Sie es mit RUN und RETURN-Taste laufen.

Auf Ihrem Bildschirm steht jetzt unter der Programm-Liste:

A = 72

Das ist also das Resultat der Zeile 4. Genauso machen wir es mit dem Punkt 5.

→ 5 PRINT "B = " B (<RETURN>)

Punkt 6 ist reiner Text, den ich in zwei PRINT-Befehle aufteile, damit der Text in zwei getrennten Zeilen steht.

→ 6 PRINT "C IST A MINUS B," (<RETURN>)

→ 7 PRINT "DAS HEISST," A "WIRD UM" B "VERRINGERT." (<RETURN>)

Zeile 6 besteht aus reinem Text - alles steht zwischen Anführungszeichen. Zeile 7 ist wieder etwas komplizierter, aber sie enthält im Grunde genommen nichts anderes als die Zeilen 4 und 5 vorher. Man muß nur die Anführungszeichen richtig abzählen. Zwischen den ersten beiden wird Text ausgedruckt, inklusive dem Komma. Dann folgt der Wert der Variablen A - ohne Anführungszeichen natürlich. Zwischen den nächsten beiden Textteilen steht wieder einsam das Stichwort B, dessen Wert der Computer suchen und hier einsetzen muß.

Auch jetzt ist es Ihnen schon erlaubt, einen vorläufigen Probelauf zu machen.

Schließlich führen wir noch Punkt 7 aus, wofür wir wieder die Methode der Zeilen 4 und 5 anwenden:

→ 8 PRINT "C =" C (<RETURN>)

Es empfiehlt sich immer, ein Programm noch mal als Ganzes aufzuLISTen. Also: Bildschirm löschen und LIST eingeben.

Wir sehen dann auf dem Bildschirm:

```
1 A = 24 * 3
2 B = 16/8
3 C = A - B
4 PRINT "A =" A
5 PRINT "B =" B
6 PRINT "C IST A MINUS B,"
7 PRINT "DAS HEISST,"A "WIRD UM" B
"VERRINGERT"
8 PRINT "C =" C
```

Alle diejenigen, welche den PRINT-Befehl mit dem Fragezeichen abgekürzt haben, werden jetzt staunen. Trotz der Abkürzerei steht im Programm-Ausdruck - auch Listing genannt - kein Fragezeichen, sondern das volle Wort »PRINT«. Das ist wieder die Tat des guten alten Editors, der flugs alles in seine rechte Bahn umlenkt und umcodiert.

Wenn Sie beim Tippen keine Fehler gemacht haben, dann dürfen Sie das Programm laufenlassen.

Wenn Sie einen Fehler entdeckt haben, dann nutzen Sie einfach den Editor aus und korrigieren Sie den Fehler direkt in der Zeile, wo er auftritt. Dabei bitte stets beachten: Nach der Reparatur nicht die Zeile verlassen, sondern zuerst RETURN drücken, damit die neue, korrigierte Zeile an die Stelle der alten falschen Zeile, die ja dieselbe Nummer hat, gespeichert wird.

So, jetzt hindert uns aber niemand mehr am RUN (und natürlich RETURN-Taste).

Auf Ihrem Bildschirm muß nun folgendes Resultat stehen:

```
A= 72
B= 2
C IST A MINUS B,
DAS HEISST, 72 WIRD UM 2 VERRINGERT
C= 70

READY.
```

Merken wir uns:

1. Innerhalb eines PRINT-Befehls können Variable und Zeichen beliebig miteinander gemischt werden. Wichtig ist nur, daß Zeichen immer innerhalb von Anführungszeichen stehen müssen.
2. Ein bereits im Progammspeicher stehendes Programm kann auf dem Bildschirm jederzeit abgeändert werden. Es gelten bei dem »Überschreiben« alle Regeln und Möglichkeiten des Editors. Eine Änderung muß mit Return abgeschlossen werden.
3. In dem Programmspeicher wird die jeweils letzte mit <RETURN> eingegebene Version einer Zeile gespeichert.
4. Zeilennummern lassen sich auf dieselbe Art und Weise verändern. Eine Zeile wird durch Eingabe lediglich der Zeilennummer, also ohne Text, gelöscht.
5. Zeilennummern brauchen nicht direkt aufeinander zu folgen. Lücken in der Zahlenfolge werden übersprungen.
6. Es empfiehlt sich, ein Programm mit Zeilennummern zu versehen, die zum Beispiel jeweils um 10 voneinander verschieden sind. Das erleichtert das spätere Einfügen von zusätzlichen Programmzeilen.
7. Eine Programmzeile kann maximal 80 Zeichen enthalten. Bei späterem Vergrößern von bestehenden Zeilen schiebt der Editor alle Zeilen automatisch nach unten, um Platz zu schaffen.

Ja, was wollen Sie machen, wenn das Programm noch immer nicht das tut, was es soll?

Als allererstes nicht verzweifeln! Tippfehler treten immer wieder auf, die der Computer in seiner Perfektion eben erbarmungslos aufgedeckt. Da er keine eigene Intelligenz hat, tut er wirklich immer nur genau das, was wir ihm eingeben - mit allen Fehlern.

Eine Fehlersuche kann oft sehr mühsam sein, besonders bei langen und komplexen Programmen.

Nun, in unserem Fall dürfte es nicht allzuschwer sein, Zeile für Zeile alle Buchstaben und Symbole durchzugehen, bis Sie den Fehler gefunden haben.

Später, in Lektion 34, werden wir noch besondere Methoden der Fehlersuche kennenlernen; jetzt möchte ich Sie noch nicht damit belasten.

Ich möchte vielmehr, daß Sie noch ein bißchen mit dem Programm spielen. Verändern Sie die Zahlenwerte in den

Zeilen 1 und 2 durch direktes Überschreiben - aber immer mit <RETURN> abschließen, sonst bleibt der alte Wert erhalten.

Sie brauchen dann nicht erneut RUN eintippen, es steht ja noch da. Fahren Sie mit dem Cursor drauf und drücken Sie die RETURN-Taste. Sofort wird das Programm wieder ausgeführt und wie mit Geisterhand erscheinen neue Zahlen an den Stellen der Variablen.

Aber Achtung!! Die Werte auf dem Bildschirm werden nur überschrieben. Wenn der neue Wert kürzer ist als der alte, dann bleibt der »längere« Rest des alten Wertes stehen; das kann zur Verwirrung führen.

Das passiert dadurch, daß dem Computer ja keine Anweisung gegeben worden ist, vor der Berechnung den Bildschirm zu löschen. Diese Anweisung kommt erst später in unser Lehrprogramm.

Beachtenswert ist, wie viele Zeichen in einer Programmzeile stehen dürfen. 40 werden Sie vielleicht sagen - aber nein: Eine mit Nummer als Programmzeile gekennzeichnete »logische« Zeile besteht aus zwei »echten« Bildschirmzeilen und enthält somit maximal 80 Zeichen.

Ein anderes »Experiment« besteht darin, die Zeilennummer einer Befehlszeile zu ändern.

→ Fahren Sie mit dem Cursor auf die 8 dieser Zeile Überschreiben Sie die 8 mit der Ziffer 9 und drücken Sie <RETURN>

Wir haben jetzt eine neue Programmzeile geschaffen, mit identischem Inhalt wie die alte Zeile 8.

→ Geben Sie LIST und RETURN ein.

Siehe da, wir haben eine neue Zeile 9, aber die alte Zeile 8 ist auch noch da, natürlich, denn wir haben sie ja durch das Überschreiben nicht gelöscht.

Das Löschen einer Zeile geht ganz einfach:

→ Tippen Sie die Nummer der Zeile, in unserem Falle eine 8, in eine freie Zeile des Bildschirms und drücken Sie <RETURN>

Wir haben also eine Zeile 8 ohne Text beziehungsweise ohne Befehle eingegeben. Da dem Rechner sein Speicherplatz sehr kostbar ist, ignoriert er eine solche nichtssagende Zeile - sie ist weg. Überprüfen Sie es mit LIST.

Jetzt haben wir ein Programm, in dem in der Reihenfolge der Zeilen eine Nummer fehlt. Was macht das? Mit RUN können Sie sich überzeugen, daß das dem Computer völlig egal ist. Er braucht nämlich nur Zeilennummern in aufsteigender Folge, Lücken überspringt er einfach.

Das bedeutet aber, daß wir unser Programm oben auch mit den Zeilennummern 10..20..30.. und so weiter hätten schreiben können. In der Tat möchte ich Ihnen diese Vorgehensweise empfehlen, da sie ein späteres Einfügen von weiteren Programmzeilen erlaubt. Wir werden das noch üben.

Wir haben ein erstes kleines Programm geschrieben. Aber sein Wortschatz ist noch sehr begrenzt - LET und PRINT. Die beiden anderen Befehle, die wir kennen, nämlich LIST und RUN, werden nur im Direkt-Modus verwendet. Bevor wir weitermachen, empfehle ich Ihnen, das alte Programm aus dem Speicher des Computers zu entfernen und ihn freizumachen für ein neues Programm.

Verwechseln Sie das bitte nicht mit dem Löschen des Bildschirms. Zur Klärung:

- Die CLR/HOME-Taste geSHIFTet löscht den Bildschirm; ein Programm im Speicher bleibt davon unberührt.
- Durch Aus- und Wiedereinschalten des Computers löschen Sie sowohl den Bildschirm als auch den Speicher des Computers, in welchem das Programm gespeichert ist. Der Computer meldet sich dann mit seinen Einschalt-Überschriften.
- Es gibt auch einen Basic-Befehl, der ein Programm aus dem Speicher hinauswirft, ohne den Bildschirm zu löschen. Er heißt

NEW

Probieren Sie es bitte aus.

→ Geben Sie eine beliebige Programmzeile (mit Nummer) ein, zum Beispiel:

```
20 PRINT"ABCDE"
```

→ löschen Sie mit der CLR-Taste den Bildschirm, geben Sie direkt LIST ein.

Jetzt erscheint die obige Zeile 20 wieder.

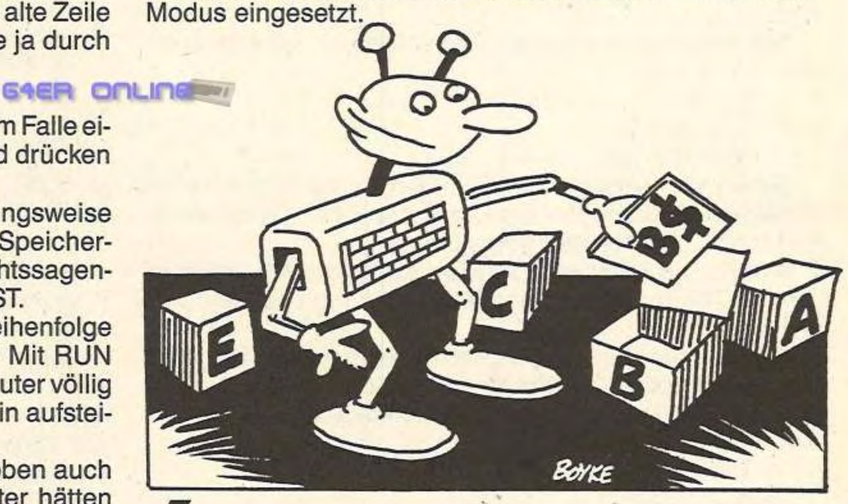
→ geben Sie direkt NEW ein und nach der RETURN-Taste wieder den Befehl LIST

Die Programmzeile 20 ist verschwunden.

Basic-Befehl Nr. 5 NEW
- macht den Programmspeicher frei für die Eingabe eines neuen Programms
- ein »altes« im Speicher sitzendes Programm wird dadurch zwar nicht gelöscht, aber es kann nach dem NEW-Befehl nur noch unter ganz bestimmten Umständen und nur mit besonderen Tricks wieder verfügbar gemacht werden

Seien Sie also vorsichtig mit diesem Befehl, immer erst überlegen, bevor Sie NEW mit der RETURN-Taste abschließen, hat er doch eine so gut wie endgültige Wirkung. Salopp ausgedrückt »löscht« er den Programmspeicher.

Auch dieser Befehl wird fast ausschließlich nur im Direkt-Modus eingesetzt.



6 Mehr über Stichwörter/Variable

Ich habe Ihnen erklärt, daß wir mit Variable die Stichwörter bezeichnen, unter denen der Computer sich Zahlenwerte merkt und unter denen sie auch wieder aus dem Speicher hervorgeholt werden können.

Des öfteren wird das alles mit Schachteln verglichen, auf die der Name einer Variablen geschrieben wird. Ein Wert, welcher der Variablen zugeordnet wird, kommt in die Schachtel mit dem richtigen Namen hinein. Eine Kopie des Wertes kann jederzeit aus der Schachtel herausgeholt werden, wenn man ihren Namen kennt. Der Wert bleibt solange in der Schachtel, bis er durch einen neuen ersetzt wird.

Es gibt mehrere Arten von Variablen, von denen wir zwei besprechen wollen.

6.1 Numerische Variable

Diesen Typ haben wir bislang verwendet. Er enthält nur Zahlen: ganze Zahlen (325), Dezimalbrüche (0.325) und negative Zahlen (-325, -0.325).

Bei der Wahl des Namens der Variablen, der vorn auf die Schachtel geschrieben wird, sind wir ziemlich frei. Er muß nur immer mit einem Buchstaben anfangen. Die nachfolgenden Zeichen können Buchstaben, Ziffern oder grafische Zeichen sein.

Wir haben bisher X, SUMME, PRODUKT etc. verwendet. Wir dürfen aber auch B23X oder F5 nehmen.

Der Länge des Variablen-Namens ist theoretisch nur durch die Anzahl der verfügbaren Zeichen in der Programmzeile eine Grenze gesetzt. Praktisch ist das aber ohne Bedeutung, da der Computer nur die ersten beiden Zeichen als Namen verwendet. So vielsagend zum Beispiel in einem Programm die Variablen-Namen »PRODUKT« und »PROFIT« sein könnten, der Computer erkennt nur »PR« und nimmt an, es handle sich um dieselbe Variable. Also Vorsicht !!!

6.2 String-Variable

So nennen wir den zweiten Typ. Hier geht es nun um Schachteln, in die wir Buchstaben, Zeichen, Wörter, ja sogar ganze Sätze bis zu einer maximalen Länge von 255 Zeichen hineingeben dürfen.

Diese Aneinanderreihung von Zeichen bezeichnen wir, wie schon erwähnt, mit »Zeichenketten« oder mit dem englischen Wort »String«, das sich wegen seiner Kürze allgemein durchgesetzt hat. Daher auch der Name »String-Variable«.

Der Name einer String-Variablen ist genauso aufgebaut wie der einer numerischen Variablen, nur muß am Ende immer das Dollar-Zeichen »$« – die geSHIFTete 4-Taste – stehen.

Der Vergleich der beiden Variablentypen sieht so aus:

```
→ 10 A = 25
  20 A$ = "ZEICHENKETTE"
  30 PRINT A
  40 PRINT A$
```

Sie sehen, selbst bei gleichem Namen, nämlich A, unterscheidet der Computer exakt zwischen der numerischen Variable A und der Stringvariable A$.

Strings und Stringvariable werden uns fortan laufend begegnen. Deswegen schlage ich zur Übung noch ein paar Beispiele vor.

Löschen Sie bitte das obige Programm mit NEW und geben Sie die folgenden Programmzeilen ein, wobei Sie genau auf Semikolons und Doppelpunkte achten müssen.

```
→ NEW
  10 T$="HOLZ"
  20 U$="FEUER"
  30 W$="7"
  40 X$="5"
  50 PRINT U$:PRINT T$
  60 PRINT U$;:PRINT T$
  70 PRINT T$;:PRINT U$
  80 PRINT W$;:PRINT X$
  90 PRINT W$ X$
```

Diese Programmzeilen ergeben auf dem Bildschirm folgendes Bild:

FEUER
HOLZ
FEUERHOLZ
HOLZFEUER
75
75

In den Zeilen 10 bis 40 werden vier String-Variable definiert, und ihnen werden zwei Wörter und zwei Ziffern zugewiesen. Die Ziffern sind nicht Zahlenwerte, sondern ebenfalls Strings, da sie zwischen Anführungszeichen stehen. Zeile 50 enthält zwei – durch den Doppelpunkt getrennte – PRINT-Befehle, die untereinander die Strings der beiden Variablen U$ und T$ ausdrucken.

Zeile 60 ist fast identisch mit Zeile 50, und doch ist ihr Resultat grundlegend verschieden, da beide Strings nebeneinander in einem Wort geschrieben werden. Der winzige Unterschied ist das Semikolon am Ende des ersten PRINT-Befehls. Wir haben das Semikolon schon kennengelernt, als Trennzeichen bei der Aneinanderreihung von mehreren numerischen Variablen innerhalb eines einzigen PRINT-Befehls.

Hier bei den Strings hat es eine entgegengesetzte Wirkung. Es klebt nämlich die Strings zweier Variablen, die in getrennten PRINT-Befehlen stehen, aneinander.

Merken wir uns:
1. Wir haben bisher zwei Typen von Variablen kennengelernt:
   – den »numerischen« Variablen weisen wir ausschließlich Zahlenwerte zu
   – den Stringvariablen, gekennzeichnet durch das $-Zeichen am Ende des Variablennamens, weisen wir ausschließlich Zeichen, Buchstaben oder ganze Folgen von ihnen zu. Diese Zeichen- und Buchstabenfolgen heißen »Zeichenketten« oder »Strings«.
   Ziffern sind auch zugelassen, nur werden sie wie Zeichen (und nicht als Zahlenwerte) behandelt.
2. Die Zuordnung von Zahlen oder Strings zum falschen Variablentyp führt generell zu einer Fehlermeldung und oft auch zum Abbruch des Programms.
3. Das Semikolon am Ende eines PRINT-Befehls bewirkt, daß der nächste PRINT-Befehl, egal wo er steht, seinen Text direkt anschließend zum vorhergehenden druckt.
4. Um ein Programm ab einer bestimmten Zeilennummer zu starten, wird diese Zeilennummer dem RUN-Befehl angehängt (RUN 600).

Ich weiß, das sieht verwirrend aus. Aber glauben Sie mir, diese »Rechtschreibregeln« werden Ihnen sehr rasch in Fleisch und Blut übergehen.

Zeile 70 ist ein weiteres Beispiel für den Alleskleber Semikolon, nur diesmal in umgekehrter Reihenfolge der Strings.

Zeile 80 zeigt Ihnen, daß die Ziffern 7 und 5 in der Tat nicht als Zahlenwerte, sondern als Strings behandelt werden. Numerische Variable werden, wie Sie sich erinnern, mit Leerzeichen für das Vorzeichen und für den Abstand ausgedruckt, Strings dagegen nicht. Und da ist es egal, ob der String zufällig eine Ziffer ist.

Zeile 90 schließlich zeigt, daß wie bei den numerischen Variablen auch mehrere String-Variable hinter einen einzigen PRINT-Befehl geschrieben werden können. Da das $-Zeichen das Ende der String-Variablen eindeutig festlegt, kann bei Strings das Semikolon weggelassen werden.

Die nächsten drei Zeilen erweitern das Programm:

```
→ 100 Y$=T$+U$
  110 Z$=U$+T$
  120 PRINT Y$:PRINT Z$
```

Die beiden String-Variablen T$ und U$ werden zu einem neuen String Y$ und in umgekehrter Reihenfolge zu Z$ zusammengebaut (Zeile 100 und 110). Zeile 120 druckt uns das Resultat aus.

Zum Schluß will ich noch meine Behauptung von oben, daß nämlich nur die beiden ersten Zeichen einer Variablen erkannt werden, beweisen.

```
→ 130 PRODUKT=25:PRINT PR
  140 PROFIT= 3:PRINT PR
  150 PRINT PRODUKT
```

Zeile 130 weist der numerischen Variablen »PRODUKT« den Zahlenwert 25 zu und druckt ihn aus, wobei die ersten beiden Buchstaben der Variablen genügen, um sie im Speicher zu finden.

Zeile 140 tut dasselbe für die Variable »PROFIT« mit dem Zahlenwert 3. Und siehe da, die Variable PR, die vorher noch 25 war, wird jetzt mit dem Wert 3 ausgedruckt. Daß wir tatsächlich in Zeile 140 der Variablen PRODUKT einen neuen Wert zugewiesen haben, obwohl wir den Variablen-Namen PROFIT gewählt haben, beweist uns Zeile 150, die unter PRODUKT dennoch die 3 ausdruckt.

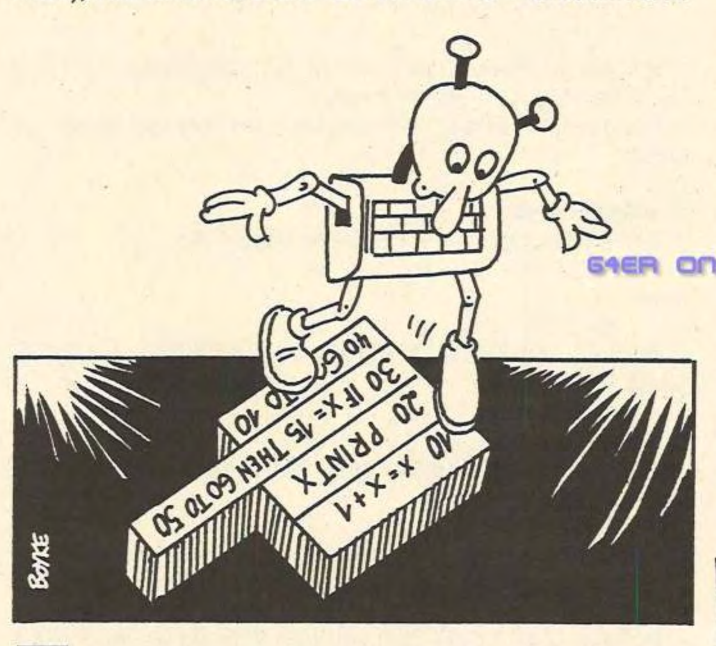
Übrigens, wenn Sie nur der Ablauf der letzten drei Zeilen des Programms interessiert, können Sie eine Eigenschaft des Befehls RUN ausnützen:

Wenn Sie

```
RUN 130
```

eingeben, läuft das Programm ab Zeile 130 los und läßt alle vorhergehenden Zeilen unbeachtet.

Wenn Sie aber nach dem RUN eine Zeilennummer angeben, die im Programm nicht vorkommt, läuft das Programm nicht los, sondern es meldet sich der Computer mit der Fehlermeldung »UNDEF'D STATEMENT« (undefined statement), was soviel heißt wie »Zeilennummer nicht definiert«.



7 Sprünge im Programm

Bislang bestand ein Programm aus einzelnen Zeilen, die der Reihe nach abgelaufen sind.

Ein Programm muß nicht so einfach sein. Es vermag auch Sprünge auf beliebige Zeilen durchzuführen.

Dazu enthält die Sprache BASIC einen Befehl, der den Computer »hüpfen« läßt, und zwar auf eine mit diesem Befehl angegebene Programmzeile. Der Befehl lautet:

GOTO (Zeilennummer)

Er darf auch in der Form GO TO geschrieben werden.

Auf deutsch bedeutet er GEHE NACH.

Durch ihn veranlaßt, springt der Computer auf die angegebene Programmzeile und fährt dort mit dem Programm fort.

Der Befehl GOTO darf auch im Direkt-Modus verwendet werden.

Ich nehme an, Sie haben noch das letzte Programm mit den Strings (Zeile 10 bis 150) im Computer. Wenn Sie jetzt direkt eintippen:

```
→ GOTO 130
```

springt der Computer auf die Zeile 130 und führt sie und die nachfolgende Zeile aus. Im Direkt-Modus wirkt er also wie der RUN-Befehl.

Im Programm-Modus sieht der Befehl genauso aus, er ist halt nur mit einer Zeilennummer versehen:

```
→ 65 GOTO 130
```

Durch den Sprungbefehl in der neuen Zeile 65 rückt das Programm auf die Zeile 130 vor und überspringt die Zeilen 70 bis 120.

Der GOTO-Befehl kann als Zieladresse aber auch eine niedrigere Zeilennummer als er selbst haben. Dann springt er im Programm zurück und bildet so eine Schleife. Geben Sie bitte ein:

```
→ 80 GOTO 60
```

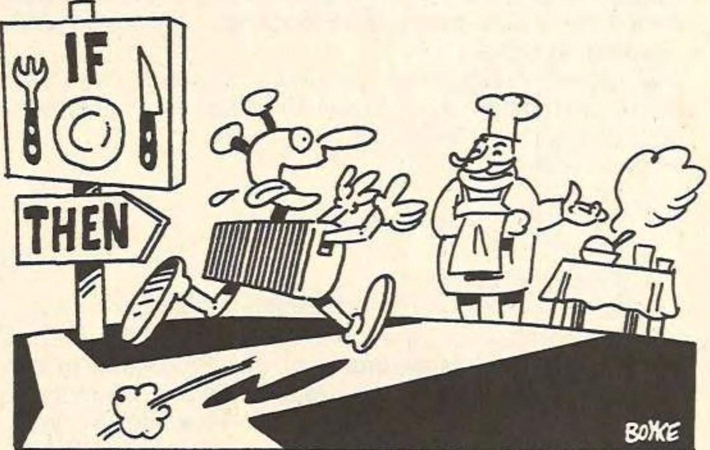
Dieser Sprungbefehl wiederholt die beiden Zeilen 60 und 70 endlos lange oder aber bis Sie die STOP-Taste drücken.

Basic-Befehl Nr. 6 GOTO

- wird mit der folgenden Schreibweise verwendet:
  GOTO (Zeilennummer)
- veranlaßt den Computer sowohl im Direkt- als auch im Programm-Modus, auf die hinter dem Befehlswort stehende Zeilennummer zu springen
- erlaubt Sprünge sowohl auf höhere, als auch auf niedrigere Zeilennummern; durch Rücksprünge können Programm-Schleifen erzeugt werden
- fehlt die Zeilennummer des GOTO-Befehls im Programm, bleibt das Programm mit der Fehlermeldung »UNDEF'D STATEMENT ERROR« mit Angabe der Zeilennummer des falschen Befehls stehen.

Die letzte Anwendung des GOTO-Befehls hat eine sogenannte Endlos-Schleife erzeugt, aus der ein Computer von allein nicht mehr herauskommt. Sie ist natürlich ziemlich sinnlos, denn außer zum Füllen des Bildschirms mit stets denselben Zeichen nützt sie uns gar nichts.

Da aber Programmschleifen sehr nützliche Werkzeuge sein können, soll ihnen eine eigene Lektion gewidmet sein.



8 Schleifen und Prüfungen

Mit Schleifen können wir zum Beispiel zählen. Das will ich Ihnen zeigen. Geben Sie ein:

```
→ NEW
  10 X=X+1
  20 PRINT X
```

Nach RUN druckt uns die Zeile 20 eine 1 aus. Am Anfang des Programms ist der Variablen X noch keine Zahl zugeordnet, also ist sie 0. In Zeile 10 wird 1 dazugezählt, was 1 ergibt.

Das ist beileibe nicht trivial, denn jetzt ergänzen wir das Programm mit:

```
→ 40 GOTO 10
```

Die dadurch gebildete Schleife wiederholt den Vorgang, und in Zeile 10 wird bei jedem Durchgang die Variable X um 1 erhöht.

Dieses kurze Programm erzeugt eine Zählschleife, die eine endlose Zahlenkolonne über den Bildschirm sausen läßt. Mit der CTRL-Taste können Sie diesen Lauf verlangsamen, mit der STOP-Taste abbrechen.

Diese endlosen Schleifen sind aber wie Autobahnen ohne Ausfahrt. Man kommt leicht drauf, und wer drauf ist, kommt nicht runter.

Wir brauchen also eine Methode, die den Aussprung aus einer Schleife ermöglicht. Dabei wollen wir aber angeben können, wann oder wo wir ausspringen.

In einer Zählschleife wäre das denkbar beim Erreichen einer bestimmten Zahl, zum Beispiel 15.

Prüfung ohne Noten

Basic kennt einen Befehl, der prüft, ob eine bestimmte, von uns festlegbare Bedingung erfüllt ist. Dann nämlich tut er, was man ihm vorgegeben hat.

Der Befehl besteht aus den beiden Wörtern

IF THEN

auf deutsch WENN DANN.

Die volle Schreibweise sieht so aus:

```
IF (Prüfbedingung) THEN (Aktion)
IF     X=3         THEN PRINT "DREI"
```

X=3 ist die Prüfbedingung, PRINT "DREI" die Aktion.

Der Befehl prüft also, ob die Prüfbedingung erfüllt ist oder nicht. Man sagt auch, er prüft, ob sie »wahr« oder »falsch« ist.

- Ist sie erfüllt, dann wird die hinter dem THEN stehende Aktion ausgeführt.
- Ist sie nicht erfüllt, dann wird ungeachtet dessen, was noch in der Zeile steht, das Programm mit der nächsten Zeile fortgesetzt.

Die Arbeitsweise sehen Sie am besten in unserem Beispiel. Fügen Sie bitte den Zeilen 10 und 20 die zwei folgenden Zeilen 30 und 50 hinzu:

```
→ 10 X=X+1
  20 PRINT X
  30 IF X=15 THEN GOTO 50
  40 GOTO 10
  50 PRINT"ENDE"
```

Also, in Zeile 10 wird X zu 1, Zeile 20 druckt die 1 aus.

Zeile 30 prüft, ob X bereits den Wert 15 erreicht hat. Dies ist nicht der Fall, deswegen geht das Programm in der nächsten Zeile – nämlich 40 – weiter. Zeile 40 springt auf Zeile 10 zurück, X wird um 1 erhöht und hat jetzt den Wert 2. Die Bedingung in Zeile 30 ist aber noch immer nicht erfüllt. Also wiederholt sich der ganze Vorgang, so lange, bis die Bedingung des IF-Befehls erfüllt ist. Erst wenn X den Wert 15 erreicht hat, tritt der THEN-Teil des Prüfbefehls in Kraft und führt das aus, was hinter dem THEN steht. In unserem Beispiel springt er mit GOTO 50 aus der Schleife heraus auf die Zeile 50.

In Zeile 50 endet das Programm mit dem Ausdruck »ENDE«.

Man muß bei der Festlegung der Prüfbedingung aufpassen, daß sie überhaupt auch auftritt. Wenn wir zum Beispiel in Zeile 10 den Wert für X jeweils um 2 erhöhen, also die Zeile 10 so schreiben:

```
→ 10 X=X+2
```

ist die Schleife wieder endlos, weil 14 oder 16 erreicht wird, aber niemals 15.

Das Problem ist lösbar mit einer Prüfung, ob X größer als 15 geworden ist. Dazu existiert ein Zeichen »>«, das mit der geSHIFTeten Punkt-Taste erzeugt wird.

```
→ 30 IF X > 15 THEN GOTO 50.
```

Jetzt wird die Zeile erst bei der Zahl 16 verlassen.

Als Prüfbedingung stehen uns mehrere mathematische und logische Ausdrücke zur Verfügung:

ist gleich	=	ist ungleich	<>
kleiner	<	Boole'sches UND	AND
größer	>	Boole'sches ODER	OR
kleiner oder gleich	<=	Negation	NOT
gleich oder größer	=>		

Die ersten sechs davon sprechen für sich selbst, die letzten drei werde ich in Lektion 31 behandeln.

Mit der Ungleich-Bedingung können wir unsere Schleife sogar noch viel eleganter gestalten:

```
→ 30 IF X <> 15 THEN GOTO 10
  40 PRINT "ENDE"
  50
```

Wir sparen Zeile 50 ein, weil die Prüfung immer erfüllt ist, bis X den Wert 15 erreicht hat.

Für den IF-THEN-Befehl sind mehrere Schreibweisen erlaubt:

1. Möglichkeit

Zeile 30 dürfen wir vereinfacht schreiben:

```
30 IF X <> 15 THEN 10
```

oder

```
30 IF X <> 15 GOTO 10
```

Anstelle von THEN GOTO kann auch THEN oder GOTO allein stehen.

2. Möglichkeit

Nach dem THEN kann auch ein anderer Basic-Befehl stehen:

```
110 PRINT X
120 IF X <> 15 THEN X=X+1
130 GOTO 110
140 PRINT "ENDE"
```

In Zeile 120 wird die Zählvariable X direkt weitergezählt, womit derselbe Effekt erzielt wird wie im Programmteil vorher (Zeile 10 bis 50).

Nur mit dem ENDE klappt es noch nicht. Sobald nämlich X=15 ist, gilt die Bedingung in Zeile 120 nicht mehr – und die Zeile 130 kommt wieder an die Reihe, was wir ja nicht wollen.

Um das zu vermeiden, muß die Zeile 120 so lauten:

```
→ 110 PRINT X
  120 IF X <> 15 THEN X=X+1:GOTO 110
  130    (entfällt)
  140 PRINT "ENDE"
```

Sie müssen also immer aufpassen und an die IF-THEN-Regeln denken:

- wenn IF »wahr« ist, wird alles hinter THEN ausgeführt
- wenn IF »falsch« ist, kommt die nächste Zeile zum Zuge

3. Möglichkeit

Mehrere Zählschleifen können unabhängig voneinander gleichzeitig ablaufen. Im folgenden Beispiel beginnt die erste Schleifenvariable X ab dem Wert 0 und wird bei jedem Durchlauf um 1 erhöht. Die zweite Schleifenvariable Y beginnt ab 28 und wird stetig um 2 reduziert:

```
→ 210 X=0:Y=28
  220 X=X+1
  230 Y=Y-2
  240 PRINT X;Y
```

Als Bedingung für das Ende wähle ich den Fall, wo X größer als Y geworden ist:

```
→ 250 IF X <  Y THEN 220
  260 PRINT "ENDE"
```

Sobald X=10 und Y=8, ist die Prüfbedingung der Zeile 250 erfüllt und das Programm bleibt stehen.

4. Möglichkeit

Mehrere Zählvariable können auch voneinander abhängig sein. Das folgende Programmbeispiel zeigt das auf einfache Weise. Obwohl diese verschachtelte Schleife keinen praktischen Wert hat, zeigt sie den Zusammenhang doch sehr eindrucksvoll:

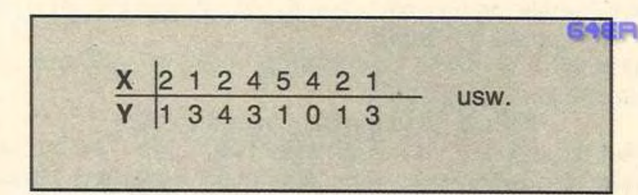
```
→ 310 X=2:Y=1
  320 X=X+Y-2
  330 Y=Y-X+3
```

In Zeile 310 stehen die Anfangswerte der Zählvariablen, Zeile 320 und Zeile 330 enthalten die Formeln, nach denen in jedem Umlauf der Schleife die Werte von X und Y »weitergezählt« werden.

Um das Resultat zu sehen, brauchen wir noch zwei Zeilen:

```
→ 340 PRINT X;Y
  350
  360 GOTO 320
```

X und Y nehmen folgende Werte an:



X	2	1	2	4	5	4	2	1	
Y	1	3	4	3	1	0	1	3	usw.

Man sieht, daß nach sechs Umläufen der Schleife die ursprünglichen Werte von X und Y wieder auftreten.

Als Prüfbedingung erlaubt uns diese Schleife eine interessante Variante. Für X kommt die Zahl 4 gleich zweimal vor, einmal bei Y=3 und noch einmal bei Y=0. Diese zweite Kombination wollen wir abfragen. Dazu nützen wir eine weitere Eigenschaft des IF-THEN-Befehls aus:

Durch Hintereinanderstellen von mehreren IF-THEN-Befehlen kann eine Mehrfach-Prüfung erzielt werden.

```
→ 350 IF X=4 THEN IF Y=0 THEN PRINT "ENDE"
```

Im dritten Umlauf der Schleife ist die erste IF-Bedingung bereits erfüllt, aber nicht die zweite. Erst nach dem 5. Umlauf bleibt die Schleife stehen.

Basic-Befehl Nr. 7 IF..THEN
- nach dem Befehlswort IF steht eine Prüfbedingung, nach dem Befehlswort THEN eine Aktionsanweisung
- IF prüft, ob die Prüfbedingung erfüllt ist, wenn ja, dann wird die Aktionsanweisung und der Rest der Zeile ausgeführt, wenn nein, läuft das Programm mit der nächsten Zeile weiter
- mehrere IF..THEN-Befehle können hintereinandergestellt eine Mehrfach-Prüfung bilden
- bei IF..THEN GOTO (Zeilennummer) kann entweder das THEN oder das GOTO weggelassen werden

Schleifen mit eigenem Befehl



Wir haben die Schleifen bis jetzt mit Hochzählen einer Schleifenvariablen und Prüfen, ob die Aussprungsbedingung schon erreicht ist, programmiert.

Basic stellt uns dafür einen sehr bequemen Befehl zur Verfügung. Er besteht aus drei Wörtern:

FOR..TO..NEXT

Auf deutsch läßt sich das übersetzen mit »Für..Bis..Der Nächste«.

Ein Beispiel soll ihn erklären. Links steht die alte Zählschleife, rechts die neue Befehlsfolge.

```
→ 10 X=X+1              10 FOR X=1 TO 10
  20 PRINT X            20 PRINT X
  30 IF X=10 THEN END
  40 GOTO 10            40 NEXT X
```

Man sieht sehr schön den Zusammenhang.

Die alte Methode der Definition einer immer um 1 höherzählenden Variablen mitsamt einer Prüfung ist jetzt in einer Zeile mit den Befehlsteilen FOR TO zusammengefaßt. Die Prüfung ist eingebaut.

Dem alten Rücksprung mit GOTO entspricht jetzt der Befehlsteil NEXT.

Die FOR-NEXT-Schleife hat immer die folgende Schreibweise:

```
FOR     Schleifenvariable = Anfangswert
TO      Endwert
STEP    Schrittweite
        .
        .
        (Programm innerhalb der Schleife)
NEXT    Schleifenvariable
```

Die Schleife

```
FOR  T=0 TO 14 STEP 2
```

setzt also die Schleifenvariable T auf Null und zählt bis 14 hoch, allerdings immer in Zweierschritten. Wenn STEP weggelassen wird, dann wird automatisch die Schrittweite 1 genommen.

Eine rückwärtszählende Schleife sieht so aus:

```
FOR T=14 TO 0 STEP -1
```

In diesem Fall muß die Schrittweite, weil negativ, immer angegeben werden.

Interessant ist auch, daß die Schrittweite ein Dezimalbruch sein kann.

```
→ 10 FOR X=1 TO 15 STEP 3.7
  20 PRINT X
  30 NEXT X
```

ergibt die Zahlen 1, 4.7, 8.4, 12.1 - also jeweils um die Schrittweite 3.7 erhöht.

Daraus ist ersichtlich, daß der NEXT-Befehl in der Schleife nicht prüft, ob die Schleifenvariable X den Endwert 15 exakt erreicht hat, sondern ob sie größer ist.

Auch der FOR-TO-NEXT-Befehl erlaubt uns, Schleifen zu »schachteln«. Was das bedeutet, zeigt uns das nächste Beispiel:

```
→ 10 FOR X=1 TO 3
  20      FOR Y=X TO X+2
  30      PRINT Y
  40      NEXT Y
  50 NEXT X
```

Die Zeilen 10 und 50 bilden die äußere Schleife. Die Zeilen 20 bis 40, die ich zur besseren Übersicht weiter eingerückt habe, bilden die innere Schleife.

Wir wollen nachvollziehen, was da abläuft.

Sie sehen, daß die innere Schleife so oft wiederholt wird, wie es die äußere Schleife vorgibt.

Der NEXT-Befehl hat noch zwei Feinheiten. Bei einer einzigen Schleife kann die Angabe der Schleifenvariablen hinter ihm wegfallen. Die folgende kleine Zeitverzögerungsschleife demonstriert das:

```
→ FOR T=1 TO 500: NEXT
```

Bei geschachtelten Schleifen stehen oft mehrere abschließende NEXT-Befehle hintereinander. Statt:

```
  40 NEXT Y
  50 NEXT X
```

kann geschrieben werden:

```
  40 NEXT Y,X
```

Dabei ist auf die Reihenfolge der Schleifenvariablen zu achten. Zuerst kommt immer die innere Variable.

Neben allen diesen geschilderten Feinheiten hat die FOR-TO-NEXT-Schleife noch einen weiteren Vorteil gegenüber der Zählschleife mit IF-THEN: Sie ist wesentlich schneller in der Ausführungszeit.

Basic-Befehl Nr. 8 FOR-TO STEP-NEXT

- hinter FOR steht die Schleifenvariable mit ihrem zugewiesenen Anfangswert. Hinter TO steht der Endwert der Schleifenvariable
- mit STEP kann die Schrittweite eingestellt werden. Sie kann auch negativ oder ein Dezimalbruch sein
- rückwärtszählende Schleifen müssen immer eine negative Schrittweite mit STEP angeben
- mehrere Schleifen können geschachtelt werden. Die innere Schleife muß vor der äußeren abgearbeitet werden
- bei NEXT kann die Schleifenvariable weggelassen werden, wenn dadurch keine Zweideutigkeiten mit anderen Schleifenvariablen entstehen
- wenn mehrere NEXT-Befehle hintereinander auftreten, können ihre Schleifenvariablen durch Kommata getrennt hinter einem einzigen NEXT stehen. Die Reihenfolge »innere Variable vor äußerer Variable« ist einzuhalten

Das folgende Beispiel soll noch einmal die Zählschleife mit IF-THEN der Version mit FOR-TO-NEXT gegenüberstellen und diesen interessanten Unterschied aufzeigen.

Mit beiden Methoden wollen wir 20 Zeilen des Bildschirms mit dem Buchstaben A füllen

```
→ 10 PRINT "(CLR-Taste)"
  20 X=X+1
  30 PRINT "A";
  40 IF X<> 20*40 THEN 20
```

Zeile 10 löscht den Bildschirm. In Zeile 30 dürfen Sie das Semikolon nicht vergessen, damit alle Buchstaben nebeneinandergeschrieben werden.

Die Prüfbedingung in Zeile 40 fragt nach, ob 20 Zeilen mal 40 Plätze gefüllt sind, eine Schreibweise, die durchaus erlaubt ist.

Schreiben Sie die andere Version direkt darunter:

```
→ 110 PRINT "(CLR-Taste)"
  120 FOR X=1 TO 20*40
  130 PRINT "A";
  140 NEXT X
```

Diese Zeilen brauche ich Ihnen nicht erklären.

Der frappierende Unterschied:
- die IF-THEN-Schleife benötigt zirka 10 Sekunden
- die FOR-TO-NEXT-Schleife benötigt nur wenig mehr als 2 Sekunden

Auf dieses Phänomen der Laufzeitunterschiede werde ich in Lektion 23 noch eingehen.

Das war wieder eine ganze Menge Information. Wer will, sollte hier vielleicht eine kleine Pause einlegen.

9 Eingabe

So, jetzt möchte ich Sie mit der Möglichkeit vertraut machen, während des Ablaufs eines Programms - also nicht schon beim Eintippen desselben - dem Computer Anweisungen in Form von Zahlen und Wörtern zu geben.

9.1. Eingabe mit Input

Bislang haben wir alle Zahlen und Wörter, die der Computer auf den Bildschirm bringen sollte, immer vorher, das heißt beim Schreiben der Programmzeilen eingegeben: als Wert-Zuweisung für eine Variable (das gute alte Stichwort), mit und ohne LET-Befehl.

Dasselbe während des Ablaufs eines Programms ermöglicht der Basic-Befehl

INPUT (gefolgt von einer Variablen)

Auf deutsch würde dieser Befehl »Eingabe« heißen. Im Direkt-Modus können wir ihn leider nicht einsetzen. Versuchen Sie es ruhig:

→ INPUT A (RETURN nicht vergessen!)

Der Computer bestraft uns mit der Fehlermeldung »ILLEGAL DIRECT ERROR«. Aber im Programm-Modus, das heißt mit einer Zeilennummer versehen, geht es. Auch das wollen wir ausprobieren:

```
→ 10 INPUT A
```

Jetzt nur noch RUN eingeben, und siehe da, der Computer druckt ein Fragezeichen und wartet mit blinkendem Cursor auf Ihre Eingaben.

Daß er wirklich wartet, merken Sie daran, daß er beharrlich immer wieder die Fehlermeldung »REDO FROM START« - was einfach »NOCH EINMAL« heißt - und danach das auffordernde Fragezeichen ausdruckt, wenn Sie einen Buchstaben oder ein Zeichen eingeben.

Er akzeptiert nur Zahlen, und außerdem noch die Leertaste, den Punkt, das Plus- und das Minuszeichen. Die RETURN-Taste bricht den Befehl ab, und das Programm fährt mit der nächsten Zeile fort.

Warum akzeptiert INPUT A nur Zahlen? Nun, was passiert bei INPUT eigentlich?

Die Zahl, die wir per Tastatur eingeben, wird der Variablen A, die hinter dem Befehlswort INPUT steht, zugeordnet, genauso wie mit dem LET-Befehl.

Mit der folgenden Programmzeile, die Sie zusätzlich zur Zeile 10 eingeben, wird diese Zuordnung bewiesen:

```
→ 10 INPUT A
  20 PRINT A
```

Die beiden Zeilen 10 und 20 zusammen ergeben ein Mini-Programm, welches nach RUN auf eine Eingabe war-

tet und diese – sofern es eine Zahl ist – auf dem Bildschirm ausdruckt.

Wir können also in der Tat mit der Tastatur innerhalb eines Programms Zahlenwerte in dasselbe eingeben, aber warum nur Zahlen?

Die Antwort ist einfach: Einer Variablen, die den Namen »A«, oder »ZAHL« oder »SUMME« hat, dürfen nur Zahlen zugeordnet werden.

Das heißt aber nicht, daß wir auf schriftliche Anweisungen, die aus Buchstaben, Zeichen oder ganzen Wörtern bestehen, verzichten müssen. Der INPUT-Befehl akzeptiert sie auch, aber nur, wenn wir eine String-Variable verwenden.

Wir können nämlich jetzt mit dem Befehl

INPUT A$

die Eingabe einer Information in Form eines Strings programmieren.

Bitte löschen Sie das letzte Programm mit NEW und geben Sie die beiden früheren INPUT-Programmzeilen neu ein, und zwar jetzt so:

```
→ 20 INPUT A$
  30 PRINT A$
```

Wenn Sie nun RUN eingeben, wartet der Computer mit dem Fragezeichen, bis Sie einen String eingeben und mit RETURN abschließen. Dann erst druckt er den String aus. Der eingegebene String A$ darf maximal 78 Zeichen enthalten.

In einem benutzerfreundlichen Programm sollte natürlich bei dem wartenden Fragezeichen von INPUT dabeistehen, welche Art von Eingabe erwartet wird. Der Benutzer des Programms sieht ja schließlich die Programmzeile nicht, in welcher der INPUT-Befehl – mit oder ohne »$« – steht.

Eine derartige Angabe können wir leicht erzeugen, indem wir eine entsprechende PRINT-Zeile vor den INPUT-Befehl der Zeile 20 setzen:

```
→ 10 PRINT"TEXT-EINGABE"
  20 INPUT A$
  30 PRINT A$
```

Als Ergebnis erhalten wir auf dem Bildschirm das Wort »TEXT-Eingabe«, darunter das Fragezeichen und wie gehabt den wartenden Cursor.

Wem das Untereinander nicht gefällt, der kann mit dem Semikolon als String-Kleber alles auf eine Zeile bringen:

```
→ 10 PRINT"TEXT-EINGABE";
```

Aber es geht noch viel eleganter !!

Der INPUT-Befehl selbst kann die Angabe enthalten. Wir nennen das einen »Kommentar« oder auf englisch ein »Prompt«.

```
→ 10
  20 INPUT"TEXT-EINGABE";A$
  30 PRINT A$
```

Zuerst wird die Zeile 10 gelöscht. Die Zeile 20 bringt dasselbe Ergebnis wie vorher die beiden Zeilen 10 und 20 zusammen.

Der Kommentar hinter dem INPUT muß immer zwischen Anführungszeichen stehen, darf maximal 38 Zeichen – auch Leerstellen zählen dazu – lang sein und muß von der Variablen (egal, ob numerisch oder String) durch ein Semikolon getrennt sein.

Und noch eine feine Einrichtung hat der INPUT-Befehl: Man darf hinter ihn mehrere Variable hängen, die allerdings alle eingegeben werden müssen. Die Schreibweise ist der des PRINT-Befehls sehr ähnlich:

```
→ 40 INPUT"4 ZAHLEN";A,B,C,D
  50 PRINT A;B;C;D
  60 INPUT"3 STRINGS";A$,B$,C$
  70 PRINT A$ B$ C$
```

Sie sehen deutlich, daß der INPUT-Befehl ein Komma zur Trennung der Variablen verlangt, der PRINT-Befehl dagegen ein Semikolon, was, wie schon erwähnt, bei String-Variablen weggelassen werden kann.

Wenn Sie nun trotz des Kommentars, vier Zahlen einzugeben, nur eine einzige eingeben und die RETURN-Taste drücken, gibt sich der Computer nicht zufrieden, sondern deutet mit einem doppelten Fragezeichen unmißverständlich darauf hin, daß noch weitere Eingaben erforderlich sind.

Die Zahlen, hintereinander eingegeben, müssen auch durch Kommata getrennt werden.

Geben Sie aber mehr Werte ein, als durch die Anzahl der Variablen hinter dem INPUT-Befehl verlangt werden, erscheint die Fehlermeldung »EXTRA IGNORED«; überzählige Werte werden von Computer also ignoriert.

Basic-Befehl Nr. 9 INPUT
- darf nur innerhalb eines Programms (also nur mit Zeilennummer) eingesetzt werden
- wird immer in der Schreibweise: INPUT (Variable) verwendet
- druckt ein Fragezeichen auf den Bildschirm und wartet auf eine Eingabe von der Tastatur, die mit RETURN abgeschlossen werden muß
- akzeptiert numerische Variable (Zahlen) und String-Variable (Text)
- weist eine falsche Zuordnung zurück, das heißt eine Zahl kann nicht für eine String-Variable und ein String nicht für eine normale Variable eingegeben werden
- mit einem einzigen INPUT-Befehl kann die Eingabe von mehreren Variablen abgefragt werden. Diese Variablen müssen durch Kommata voneinander getrennt sein
- werden nicht alle geforderten Variablen eingegeben, fragt INPUT mit einem doppelten Fragezeichen nach den fehlenden Werten
- werden mehr Werte als gefordert eingegeben, meldet der Computer dies mit einer Fehlermeldung »EXTRA IGNORED«
- bei String-Eingaben dürfen maximal 78 Zeichen eingegeben werden
- INPUT kann mit einem Kommentar versehen werden, den es vor dem Fragezeichen ausdruckt. Dieser Kommentar muß so geschrieben werden: INPUT "KOMMENTAR";(Variable)
- der Kommentar darf maximal 38 Zeichen lang sein

Diese Eingabemethode für Strings funktioniert gut, aber sie hat einige Eigenheiten und Einschränkungen: Dazu ein paar Beispiele:

Starten Sie die Zeilen 10 und 20 mit RUN. Wenn Sie hinter dem blinkenden Fragezeichen eingeben:

```
WERT:ZAHL  oder  WERT,ZAHL
```

erscheint eine Fehlermeldung und der Ausdruck:

EXTRA IGNORED
WERT

Der Text ab dem Komma beziehungsweise ab dem Doppelpunkt wird unterschlagen.

Die Eingabe:

```
"WERT" IST
```

führt zur Fehlermeldung REDO FROM START.

Bei einer Eingabe:

```
"DER WERT
```

wird das Anführungszeichen nicht geschrieben.

Dagegen wird die folgende Eingabe akzeptiert:

```
DER "WERT" IST
```

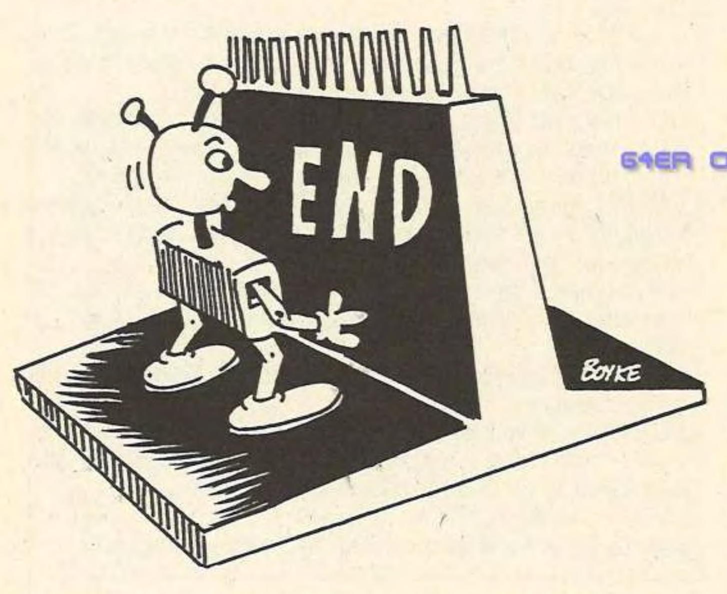
Sie sehen, man muß aufpassen.

Merken wir uns:
1. Komma und Doppelpunkt dürfen im einzugebenden Text nicht vorkommen
2. einen Zeilensprung mit RETURN einzugeben geht nicht, da diese Taste den INPUT-Vorgang beendet
3. der Text darf nicht mit einem Anführungszeichen anfangen
4. der Text – mit Leerstellen – darf nicht länger als 78 Zeichen sein
5. der Eingabevorgang kann nicht mit der STOP-Taste abgebrochen werden.

Der INPUT-Befehl in Verbindung mit der IF-THEN-Abfrage ist eine wichtige Möglichkeit zur Steuerung und Beeinflussung von Programmen durch den Benutzer. Wir werden sie immer wieder antreffen. Zwei kleine Beispiele sollen diese Befehlskombination verdeutlichen:

In einem Spielprogramm soll der Spieler eine Zahl zwischen 5 und 10 eingeben. Das geht so:

```
→ 10 PRINT"GEBEN SIE EINE ZAHL ZWISCHEN
     5 UND 10 EIN"
  20 INPUT "ZAHL (5-10)";Z
  30 IF Z <  5 THEN 20
  40 IF Z > 10 THEN 20
  50 PRINT Z
```



Jede eingegebene Zahl Z, die kleiner als 5 oder größer als 10 ist, wird durch die Zeilen 30 und 40 durch Rücksprung auf den INPUT-Befehl zurückgewiesen.

Die Prüfvariable kann auch ein String sein. Bitte ergänzen Sie das Programm mit den folgenden Zeilen:

```
→ 60 INPUT "JA ODER NEIN";A$
  70 IF A$="JA" THEN GOTO 100
  80 IF A$="NEIN" THEN GOTO 110
  90 PRINT "FALSCHE EINGABE":GOTO 60
  100 PRINT "JA"
  110 PRINT "NEIN"
```

Das wollen wir jetzt zeilenweise analysieren.

Zeile 60 ist der INPUT-Befehl, der uns im Kommentar auffordert, entweder »JA« oder »NEIN« einzugeben.

Zeile 70 prüft, ob wir »JA« eingegeben haben. Ist das der Fall, dann springt sie auf Zeile 100, und wir erhalten den Ausdruck »JA«. Haben wir nicht das »JA« eingegeben, geht das Programm in der nächsten Zeile (80) weiter.

In Zeile 80 wird geprüft, ob wir »NEIN« eingegeben haben. Wenn das zutrifft, springt sie auf Zeile 110, und das Wort »NEIN« wird ausgedruckt. Haben wir aber »NEIN« auch nicht eingegeben, sondern irgend ein anderes Wort (String), dann geht das Programm in der nächsten Zeile (90) weiter.

Ich hoffe, Sie sehen wieder das typische Muster des IF..THEN-Befehls.

In Zeile 90 schließlich wird die Ermahnung ausgedruckt, doch nur mit »JA« oder »NEIN« zu antworten, und mit dem Rücksprung GOTO 60, mit dem Doppelpunkt als zweitem Befehl der Zeile 90 angehängt, wird eine neue Eingabe verlangt.

Einen kleinen Fehler hat das Programm noch. Wenn Sie es mit RUN von Anfang an oder mit RUN 60 nur ab der INPUT-Zeile starten und gleich mit »JA« antworten, kommt zuerst Zeile 100 zum Zuge, aber gleich danach auch Zeile 110, und wir erhalten beide Ausdrucke, »JA« und »NEIN«.

Zeile 110 können wir in diesem Fall ausblenden, entweder durch eine Überbrückung mit GOTO (irgendwohin) oder mit einem neuen Basic-Befehl, der das Programm mit Zeile 100 beendet. Er heißt END.

Endstation mit END

Ändern Sie bitte Zeile 100 ab:

```
→ 100 PRINT"JA":END
```

Jetzt klappt es.

Der END-Befehl beendet also die Ausführung eines Programms.

Basic-Befehl Nr. 10 END

Dieser Befehl steht für sich allein oder nach einem IF..THEN-Befehl als Aktionsanweisung.

Er beendet sofort ein Programm und der Computer meldet sich mit READY und blinkendem Cursor.

9.2. Eingabe mit GET

In Basic gibt es noch einen zweiten Befehl, der im Prinzip das gleiche wie INPUT macht. Nur im Detail der Ausführung unterscheidet er sich vom Kollegen INPUT. Diese Unterschiede aber machen ihn zu einer wertvollen Alternative. Der Befehl heißt

GET (Variable)

was soviel bedeutet, wie »holen, bekommen«.

Der GET-Befehl holt also einen Wert und weist ihn der hinter ihm stehenden Variablen zu. Diese Variable kann eine numerische oder eine String-Variable sein.

Unterschied zwischen GET und INPUT
- INPUT wartet, bis eine Eingabe mit der RETURN-Taste abgeschlossen ist
- GET wartet nicht, sondern prüft lediglich, ob eine Taste gedrückt worden ist.
- INPUT erlaubt die Eingabe bis zu 78 Zeichen
- GET holt immer nur 1 Zeichen
- INPUT meldet sich mit Fragezeichen und Cursor
- GET meldet sich auf dem Bildschirm nicht

Die Frage stellt sich, wie der GET-Befehl prüfen kann, ob eine Taste gedrückt worden ist, wenn er nicht wartet. Daß er gerade in dem kurzen Zeitpunkt prüft, in dem zufällig der Tastendruck stattfindet, ist mehr als unwahrscheinlich.

Und in der Tat, wenn nicht das Zeichen jeder gedrückten Taste in einen Pufferspeicher käme, ehe es weiterverwendet wird, hätte der GET-Befehl keine Chance, jemals eine Eingabe zu erwischen.

Der Tastaturpuffer

Diesen Speicher des Computers, in dem die Zeichen gedrückter Tasten erst einmal zwischengelagert werden, möchte ich Ihnen kurz zeigen.

Dazu brauchen wir eine Zählschleife, wie wir sie beim GOTO-Befehl erstmals angewendet haben. Hier brauche ich sie, um eine sogenannte Zeitverzögerungsschleife zu bauen. Diese tut das, was ihr Name sagt: sie zählt eine gewisse Zeit still vor sich hin und verzögert so den nächsten Programmschritt. Entfernen Sie bitte alle Programmzeilen mit NEW

```
→ NEW
  30 X=X+1
  40 IF X <> 500 THEN 30
```

Mit RUN gestartet, läuft die Schleife in diesen beiden Zeilen 500mal durch, bis sich der Editor mit READY und Cursor wieder meldet.

Diese Zeit nützen wir, um so viele Tasten wie möglich hintereinander zu drücken. Natürlich sehen wir nichts auf dem Bildschirm, während das Programm der Zeitschleife läuft. Die Behauptung, daß die gedrückten Zeichen in einen Pufferspeicher kommen, bewahrheitet sich nach dem Ende der Zeitschleife.

Nach dem READY holt nämlich der Editor alle Zeichen aus dem Tastaturpuffer und druckt sie aus. Da dieser Puffer nur 10 Zeichen speichern kann, sehen Sie nur 10 Zeichen, selbst wenn Sie es geschafft haben, mehr einzugeben. Also nochmal:

→ RUN eingeben
möglichst viele Tasten drücken, aber nicht gleichzeitig nach dem READY die Zeichen zählen

In diesem Tastaturpuffer schaut also der GET-Befehl nach, ob vorher ein Tastendruck ein Zeichen darin untergebracht hat. Wenn ja, dann verwendet er es, falls nein, kommt die nächste Programmzeile dran.

Um das zu zeigen, erweitern wir das obige Programm um vier weitere Zeilen:

```
→ 10 GET A
  20 PRINT A
  30 X=X+1
  40 IF X <>  100 THEN 30
  50 X=0
  60 GOTO 10
```

In Zeile 10 schaut der GET-Befehl im Tastaturpuffer nach einem Zeichen. Was immer er findet, druckt Zeile 20 auf den Bildschirm. Eine 0 signalisiert, daß der Puffer leer war.

Die Zeilen 30 und 40 zählen von 1 bis 100, wie beim Versteckspielen, und machen dann nach 100 Zähleinheiten in Zeile 50 weiter, in der die Zählvariable X wieder auf Null gesetzt wird.

Zeile 60 springt zurück auf den GET-Befehl und alles fängt mit X=0 wieder von vorn an. Das Resultat ist eine langsam fortschreitende senkrechte Kolonne von Ziffern links am Bildschirm. Die Geschwindigkeit ist durch die Prüfzahl in Zeile 40 einstellbar.

Während der Zählschleife haben wir nun Gelegenheit, Zahlen einzugeben, die als einzelne Ziffern vom GET-Befehl geholt und durch Zeile 20 ausgedruckt werden.

Die Zeitschleife der beiden Zeilen 30 und 40 können wir natürlich auch mit der FOR-TO-NEXT-Schleife schreiben. Da diese Schleifenart aber viel schneller abläuft, muß die Zählvariable 10mal so groß sein.

```
→ 30 FOR X=0 TO 1000: NEXT
  40   (entfällt)
  50   (entfällt)
```

Wenn Sie einen Buchstaben eingeben, bricht das Programm mit einer Fehlermeldung ab.

Um Buchstaben und Zeichen eingeben zu können, müssen wir die numerische Variable A in eine String-Variable A$ umwandeln – natürlich in beiden Zeilen 10 und 20.

Jetzt holt der GET-Befehl Buchstaben – aber auch Ziffern! Diese werden jedoch als Zeichen (einstellige Strings) und nicht als Zahlenwerte behandelt.

Basic-Befehl Nr. 11 GET
- der Befehl GET (Variable) holt eine Zahl oder ein Zeichen aus dem Tastaturpuffer und weist es der Variablen zu
- die Variable kann eine numerische oder eine String-Variable sein
- entspricht das Zeichen bzw. die Zahl nicht dem Variablentyp, wird mit Fehlermeldung abgebrochen
- GET wartet nicht. Falls der Puffer leer ist, weist er der Variablen den Wert 0 beziehungsweise einen sogenannten »Nullstring« (kein Leerzeichen, einfach gar nichts) zu
- im Gegensatz zu INPUT sind bei der Eingabe mit GET alle Zeichen erlaubt, auch Komma, Doppelpunkt und Semikolon

Jetzt will ich Ihnen eine echte Anwendung des GET-Befehls zeigen. Wir haben in Lektion 9 bei der INPUT-Eingabe einen Programmteil geschrieben, um den Benutzer zu fragen, ob er mit JA oder NEIN antworten will.

Mit dem INPUT-Befehl ging das so:

```
→ 10 PRINT"BEGINN"
   .
   .      (Programm)
   .
  320 INPUT"NOCH EINMAL (J/N)";A$
  340 IF A$="J" THEN 10
  350 IF A$ <>" N" THEN 320
  360 PRINT"AUF WIEDERSEHEN"
```

Ab Zeile 10 beginnt das Programm.
Zeile 320 fragt nach J(a) oder N(ein).
Zeile 340 prüft, ob die Eingabe "J(a)" ist, im Fall »wahr« springt das Programm an den Anfang zurück.

Im Fall »falsch« kommt Zeile 350 an die Reihe mit der Prüfung, ob die Eingabe ein ungültiges Zeichen ist – alle Zeichen außer »NEIN« sind ungültig. Leider ist die Eingabe »NEIN« auch falsch, denn wir prüfen ja nur auf »N«. Das ist ein Nachteil des INPUT-Befehls.

Im Fall »wahr« beziehungsweise »J« wird der INPUT-Befehl wiederholt. Wenn es ein »N« war, verabschiedet sich das Programm in Zeile 360.

Dasselbe machen wir jetzt mit dem GET-Befehl. Die Zeilen 10, 340, 350 und 360 bleiben unverändert.

```
→ 10 PRINT"BEGINN"
   .
   .      (Programm)
   .
  320 PRINT"NOCH EINMAL (J/N)?"
  330 GET A$:IF A$=""THEN 330
  340 IF A$="J"THEN 10
  350 IF A$ <>"N" THEN 320
  360 PRINT"AUF WIEDERSEHEN"
```

Zeile 320 druckt die Anfrage aus, die oben hinter dem INPUT-Befehl steht. In Zeile 330 schaut der GET-Befehl im Tastaturpuffer nach, ob er ein Zeichen enthält. Mit IF A$= " " prüft er, ob der Puffer leer ist. Die doppelten Anführungszeichen ohne Zwischenraum repräsentieren den bei der Befehlsbeschreibung genannten »Nullstring«, das heißt, im Puffer war kein Zeichen. Deswegen springt diese

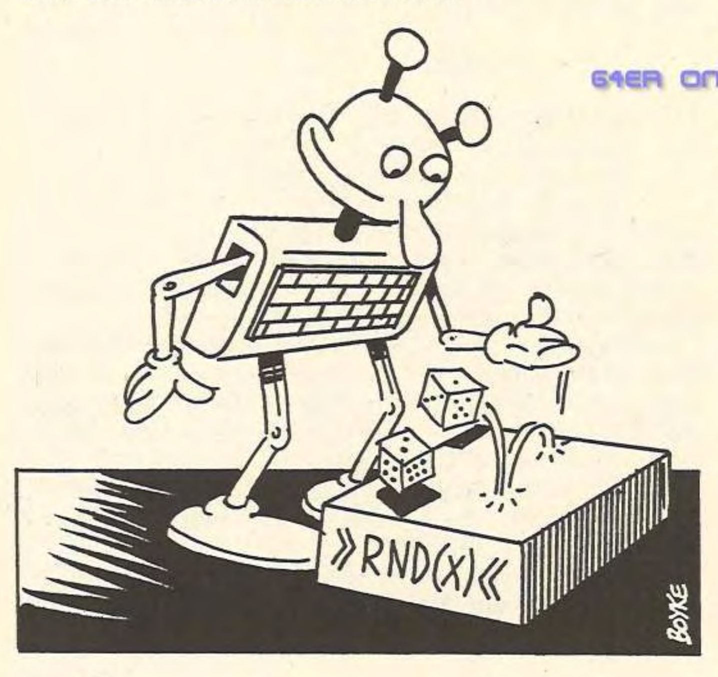
Zeile 330 auf sich selbst zurück und verharrt in dieser Schleife so lange, bis die Prüfbedingung des Nullstrings nicht mehr erfüllt ist. Das ist sie, sobald irgendeine Taste gedrückt worden ist. Die restlichen Zeilen prüfen genauso wie im Beispiel vorher. Der Hauptunterschied ist der, daß der GET-Befehl auch »NEIN« akzeptiert, prüft er doch immer nur 1 Zeichen, nämlich das erste – er würde natürlich auch »NAME« akzeptieren. Außerdem reagiert er sofort und nicht erst nach dem Drücken der RETURN-Taste. Prinzipiell möchte ich sagen, daß der GET-Befehl »intelligenter« ist als INPUT. Er bietet mehr Flexibilität und auch mehr Eleganz – eine Eigenschaft, in der sich oft gute und schlechte Programme unterscheiden.

Merken wir uns:
1. der GET-Befehl wartet nicht
2. wenn er auf eine Eingabe warten soll, muß er mit einer »Nullstring-Abfrage« in eine einzeilige Warteschleife gelegt werden:

```
20 GET A$:IF A$="" THEN 20
```

Es könnte sein, daß der eine oder andere Leser jetzt seufzt und sich fragt, wie denn mit allen diesen Befehlen jemals ein Programm zusammenstellbar sei.

Ich habe in der Tat vor, so schnell wie möglich mit Ihnen ein erstes sinnvolles und für Sie reizvolles Programm zu entwickeln. Natürlich wird es ein Spiel sein. Wir brauchen dazu aber noch zwei weitere Befehle.



10 Zufallszahlen und ganze Zahlen

Der Zufall im Computer ist eigentlich ein Widerspruch in sich, da doch im Computer alles durch die Anweisungen des Programms fest vorgegeben ist. Und doch gibt es etwas Zufallähnliches.

Es ist sowohl bei mathematischen Anwendungen als auch bei Spielen oft erforderlich, von einer nicht vorher bekannten, völlig zufällig entstandenen Zahl auszugehen. Wie könnte sie anders heißen als »Zufallszahl«.

Basic hat einen Befehl, der eine derartige Zufallszahl erzeugt. Er heißt: RND (X)

Der Name ist abgeleitet aus RANDOM, was auf deutsch »zufällig« bedeutet.

Das X, das in Klammern hinter dem Befehlswort steht, wird Argument genannt. Das X kann drei Werte haben:
- eine positive Zahl (egal, welcher Wert)
- eine negative Zahl
- die Zahl 0

Bevor wir darauf eingehen, möchte ich Ihnen die Wirkungsweise von RND zeigen, wie immer am besten durch ein Experiment.

```
→ 10 A=RND(1)
  20 PRINT A
  30 GOTO 10
```

Diese drei Zeilen bilden eine Schleife, die Ihnen in einem Zahlenstreifen die Werte zeigt, die RND(1) erzeugt. Wir sehen vielstellige Zahlenwerte, die abgerundet zwischen 0 und 1 liegen.

Ab und zu taucht in der Zahlenkolonne eine Zahl in eigenartiger Schreibweise auf, mit einer Ziffer vor dem Dezimalpunkt und einem »E–« gefolgt von einer Zahl. Das ist die sogenannte wissenschaftliche Schreibweise, die im Handbuch von Commodore genauer erklärt wird. Hier bitte ich Sie, diese Zahlen ganz einfach zu ignorieren.

Zurück zu den Zahlen, die von RND erzeugt werden.

Sie werden durch eine komplizierte mathematische Formel errechnet, die zwar keine absolute Zufälligkeit garantiert, ihr aber sehr nahekommt.

Sie können sich vorstellen, daß die Zufälligkeit ganz wesentlich von der oben genannten Anfangszahl abhängt. Und gerade sie wird durch das Argument (X) beziehungsweise durch die drei Arten des Arguments bestimmt.

Das positive X beginnt nach dem Einschalten des Computers immer mit derselben Anfangszahl. Bei X=0 wird als Anfangszahl der Stand der inneren Computeruhr genommen. Bei einer negativen Zahl bildet diese selbst den Anfangswert.

Ich empfehle Ihnen, den Wert 0 zu nehmen. Mit 0 als Argument von RND will ich die drei Zeilen von oben neu schreiben, diesmal als sogenannten »Einzeiler«. Darunter versteht man ein Programm, welches in eine einzige Zeile paßt. Glauben Sie mir, in dieser Art sind schon ganze Kunstwerke veröffentlicht worden.

Basic-Befehl Nr. 12 RND(X)
- wird für X die Zahl 0 oder irgendeine positive Zahl genommen, erzeugt dieser Befehl eine zirka achtstellige Zahl zwischen Null und Eins. Ihr jeweiliger Wert ist sozusagen zufällig.

Diesen Anspruch erhebe ich nicht, nur den der Vereinfachung.

```
→ 10 PRINT RND(0):GOTO 10
```

Die Zeile braucht unbedingt eine Zeilennummer, damit der GOTO-Befehl eine Schleife bilden kann.

Zufallszahlen haben wir jetzt, aber leider in der Form eines Dezimalbruches. Wenn wir in einem Programm zum Beispiel würfeln wollen, brauchen wir Zufallszahlen von 1 bis 6. Dezimalbrüche helfen uns da nicht viel. Wir brauchen also eine Möglichkeit, diese Dezimalbrüche in ganze Zahlen umzuwandeln.

Auch dafür hat Basic einen Befehl. Er lautet:

INT

Das ist die Abkürzung von INTEGER, was auf deutsch »ganze Zahl« bedeutet.

Der Dezimalbruch, der mit INT in eine ganze Zahl verwandelt werden soll, wird in Klammern hinter den Befehl geschrieben. Der Befehl ist im Direktmodus verwendbar:

```
→ PRINT INT(25.38)
```

Wir erhalten die Zahl 25.

INT macht es sich also leicht - es schneidet einfach alle Zahlen hinter dem Dezimalpunkt weg.

Es ist auch erlaubt, genau wie beim PRINT-Befehl eine Formel mit INT zu versehen:

```
→ PRINT INT(2.8908*567.8)
```

Das Ergebnis ist 1641.

Basic-Befehl Nr. 13 INT(A)
- wandelt einen Dezimalbruch A in eine ganze Zahl um
- A kann nicht nur ein Dezimalbruch, sondern eine mathematische Formel oder ein komplizierter mathematischer Ausdruck sein
- die Klammern dürfen nicht weggelassen werden

Diesen Befehl INT wenden wir jetzt an, um aus den unerwünschten Dezimalbrüchen des RND-Befehls ganze Zahlen zu machen.

Zuerst nehme ich der Klarheit halber die 3zeilige Version, welche die Zufallszahl einer Variablen zuordnet:

```
→ 10 A=RND(0)
  20 PRINT INT(A)
  30 GOTO 10
```

Der Unterschied zu vorher liegt hier in der Zeile 20. Das Resultat nach RUN ist enttäuschend, aber verständlich, denn der INT-Befehl macht aus Dezimalbrüchen, die kleiner als 0 sind, nur eine 0.

Zeile 20 muß verbessert werden, indem der Dezimalpunkt von A nach rechts verschoben wird. Das erreichen wir durch die Multiplikation mit 10, 100, 1000 und so weiter.

```
→ 20 PRINT INT (A*10)
```

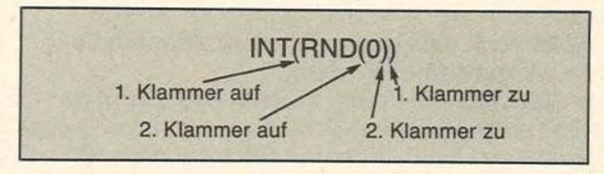
liefert Zufallszahlen von 0 bis 9. Den Bereich 0 bis 99 erhalten wir durch:

```
→ 20 PRINT INT (A*100)
```

Als Einzeiler sieht diese letzte Version so aus:

```
→ 10 PRINT INT(RND(0)*100):GOTO 10
```

Beachten Sie bitte die Verschachtelung der Klammern:



Wenn eine Klammer fehlt oder falsch ist, erhalten wir die Fehlermeldung »SYNTAX ERROR«. Am besten ist es, Sie überprüfen, daß die Anzahl der geöffneten und geschlossenen Klammern immer gleich ist. Im Merksatz finden Sie ein Kochrezept, mit dem Sie Zufallszahlen innerhalb des gewünschten Bereiches erzeugen können.

Merken wir uns:
Die Formel:

```
→ A = INT(RND(0)*X)+Y
```

erzeugt ganze Zahlen innerhalb des Zahlenbereiches Y bis Y+X-1.
Beispiel:
Zufallszahlen von 20 bis 90 machen Y zu 20 und erfordern ein X, das aus der Formel Y+X-1=90 errechnet wird. Anders geschrieben lautet diese Formel X=90-Y+1. Das ergibt in unserem Fall X=71.

```
→ 10 PRINT INT(RND(0)*71)+20
```

Im Zweifelsfall lohnt es sich immer, die Formel mit einer derartigen Zeile 10 auszuprobieren.



11 Ein vollständiges Programm

Wir sind gerüstet für ein erstes Programm. Es heißt »ZAHLEN RATEN« und ist aus dem Buch »Computerspiele und Knobeleien« von Rüdiger Baumann abgeleitet. Ein ähnliches Zahlenratespiel finden Sie übrigens auch im Commodore 64-Handbuch (Seite 51 f.)

Aufgabe:

Eine vom Computer zufällig gewählte Zahl zwischen 1 und 100 soll in möglichst wenigen Versuchen erraten werden. Nach jedem Rateversuch gibt der Computer einen Hinweis »zu groß« oder »zu klein«. Wurde richtig geraten, nennt der Computer die Zahl der Versuche und stellt anheim, das Spiel noch einmal zu versuchen oder zu beenden.

11.1. Die Planung eines Programms

Ich habe ganz am Anfang dieses Kurses erwähnt, daß Basic dazu verleitet zu »hacken«. In der Tat führt die an sich sehr positive Eigenschaft dieser Programmiersprache, nämlich schon kleinste Bruchstücke eines Programms eingeben und dann gleich ausprobieren zu können, leicht zu einem sogenannten »Spaghetti-Programm«. Die einzelnen Bruchstücke kann man nämlich stehenlassen und die anderen Teile irgendwie dazwischenschieben oder mit GOTO irgendwo anhängen. Das Resultat ist aber am Schluß ein wilder Haufen von Befehlszeilen, die zwar funktionieren, aber nicht mehr nachvollziehbar sind.

Es ist daher anzuraten, sich den Ablauf eines Programms vorher zu überlegen, was ja nicht schwer ist, weil ein Computer nie mehrere Sachen gleichzeitig macht, sondern alle streng hintereinander abwickelt. So einen Ablauf kann man schön aufs Papier malen, und das will ich Ihnen jetzt zeigen.

Zuerst schreiben wir ganz einfach alle Vorgänge der Reihe nach auf. Wenn die Reihenfolge uns nicht paßt, wird sie einfach abgeändert.

Am Ende kann es so aussehen:

1. Bildschirm löschen
2. Überschrift schreiben
3. Zufallszahl zwischen 1 und 100 erzeugen
4. Zähler für Zahl der Versuche auf Null stellen
5. Anweisungen an den Spieler
6. Eingabe einer geratenen Zahl

7. Zähler der Versuche um 1 erhöhen
8. Prüfung der geratenen Zahl auf:
   - zu groß: Hinweis geben und neuer Versuch
   - zu klein: Hinweis geben und neuer Versuch
   - richtig: Zahl der Versuche ausdrucken
9. Frage, ob noch einmal
   - ja: zurück zum Anfang
   - nein: Verabschiedung, Ende

Alles, was jetzt noch zu tun bleibt, ist, die einzelnen Punkte der Reihe nach mit den Befehlen, die wir kennen, als Programmzeilen zu schreiben.

Übrigens: Ich schreibe zuerst nur die »aktiven« Programmzeilen. Alle anderen Teile des Programms, die nur der Lesbarkeit dienen, folgen später. Dadurch erklären sich auch die Sprünge in den Zeilennummern, die ich Sie bitte zu beachten.

11.2. Die Durchführung

Punkt 1 und 2:

Den Bildschirm löschen wir mit einem PRINT-Befehl, hinter dem wir nach dem ersten Anführungszeichen die geSHIFTete CLR/HOME-Taste drücken – was bekanntlich im Direktmodus den Löschvorgang auslöst. Im Listing erscheint dann ein invertiertes Herz. Was es damit auf sich hat, erkläre ich im Anschluß an das Programm.

```
→ 10 PRINT" {SHIFT CLR/HOME}"
  20 PRINT"****** ZAHLEN RATEN ******
```

Punkt 3:

Der Bereich der Zufallszahl soll zwischen 1 und 100 liegen. Entsprechend dem dafür angegebenen Kochrezept INT(RND(0)*X)+Y errechnen wir wieder mit der unteren Grenze 1=Y und der oberen Grenze X+Y-1=100 das uns noch fehlende X:

X+1-1=100

X=100

Dann ergibt sich für die Zufallszahl »Z« die folgende Programmzeile:

```
→ 70 Z=INT(RND(0)*100)+1
```

Punkt 4:

Der Zähl-Variablen für die Anzahl der Versuche geben wir den Namen »V«. In Zeile 80 wird sie auf Null gesetzt.

```
→ 80 V=0
```

Punkt 5:

Die Anweisungen an den Spieler stehen in Zeile 120 und 130. Sie können sie in Listing 1 – so nennen wir den kompletten Ausdruck eines Programms – in allen Einzelheiten nachlesen.

Sie können die Anweisung aber Ihrem Geschmack entsprechend auch anders schreiben oder anders anordnen.

Punkt 6:

Die Aufforderung zur Eingabe einer Zahl und die Eingabe selbst besorgt uns wieder der INPUT-Befehl

```
→ 140 PRINT
  150 INPUT"WELCHE ZAHL IST ES";R
```

Vor dem INPUT-Befehl drucken wir erst eine Leerzeile aus, damit es besser aussieht.

Der einzugebenden Zahl stellen wir die numerische Variable »R« zur Verfügung, diesmal ohne $-Zeichen, da wir ja Zahlen eingeben wollen.

Punkt 7:

Sobald die Zahl eingegeben ist, muß die Variable V des Versuche-Zählers um 1 erhöht werden.

```
→ 160 V=V+1
```

Punkt 8:

Jetzt kommt die Prüfung mit IF THEN. Ich habe sie wegen der nachfolgenden Hinweise ZU GROSS, ZU KLEIN und RICHTIG in 3 Gruppen zu je 2 Stufen aufgebaut.

Das Grundprinzip dieser Prüfung liegt im Vergleich der beiden Zahlen, nämlich der geheimen Zahl des Rechners »Z« und der eingegebenen Zahl »R« des Raters. Zeile 200 prüft auf »größer«, Zeile 210 auf »kleiner«, Zeile 220 auf »gleich«.

```
→ 200 IF R < Z THEN PRINT R;:GOTO xxx
  xxx PRINT"IST ZU GROSS":GOTO 140
```

Die Zeilennummer xxx tragen wir dann nach, sobald klar ist, wie viele Zeilen noch dazwischenliegen.

Wichtig in Zeile 200 ist das Semikolon nach der geratenen Zahl R. Es klebt wieder einmal den nachfolgenden Satz der Zeile xxx direkt hinter den Wert der Zahl R, so daß auf dem Bildschirm der Satz erscheint:
»(Zahl R) IST ZU GROSS«.
Genauso ist es mit den beiden anderen Prüfungen:

```
→ 210 IF R > Z THEN PRINT R;:GOTO yyy
  yyy PRINT"IST ZU KLEIN":GOTO 140
  220 IF R = Z THEN PRINT R;:GOTO zzz
  zzz PRINT"IST RICHTIG"
```

Aus der Reihenfolge wird jetzt klar, daß xxx die Zeile 230 ist, entsprechend yyy=240 und zzz=250. Bitte tragen Sie dies nach.

Die beiden falschen Ergebnisse in Zeile 230 und 240 führen also zum Rücksprung auf eine neue Eingabe, die ab Zeile 140 beginnt.

Ein richtiges Ergebnis führt in die nächsthöhere Zeile 260 weiter.

In Zeile 260 und 270 drucken wir jetzt den letzten Stand des Versuche-Zählers V aus, eingebettet in Text, von dem er durch zwei Semikolons getrennt werden muß:

```
→ 260 PRINT"SIE HABEN";V;"VERSUCHE ";
  270 PRINT"BENOETIGT"
```

Ich habe mit Absicht den Text, der ja leicht in eine Zeile gepaßt hätte, in zwei Zeilen getrennt. Erstens verbessert er das Format des Listings. Zweitens aber wollte ich Ihnen zeigen, wie man trotzdem den Text in eine Zeile ausgedruckt erhält. Das Geheimnis liegt wieder im Semikolon nach dem letzten Wort der Zeile 260; zusätzlich aber ist die Leerstelle vor dem abschließenden Anführungszeichen wichtig. Ohne sie würden die beiden Wörter »VERSUCHE« und »BENOETIGT« zu einem einzigen Wort verklebt werden.

Punkt 9:

Es bleibt noch die Frage nach einer Wiederholung. Das machen wir mit dem GET-Befehl.

Zeile 320 stellt die Frage, in Zeile 330 wartet der GET-Befehl mit einer Nullstring-Schleife auf eine gedrückte Taste. Zeile 340 prüft auf »J(a)« und springt zurück auf den Anfang, Zeile 350 prüft auf ungültige Eingaben mit Wiederholung der Aufforderung, und wenn »N(ein)« eingegeben ist, schließt Zeile 360 das Programm mit höflicher Verabschiedung.

```
→ 320 PRINT"NOCH EINMAL (J/N) ?"
  330 GET A$:IF A$="" THEN 330
  340 IF A$="J"THEN 10
  350 IF A$ <> "N" THEN 320
  360 PRINT:PRINT:PRINT"AUF WIEDERSEHEN"
  370 END
```

So, das war doch nicht schwer, oder?
Alles, was passieren kann, sind Tippfehler beim Eingeben.

Im nachfolgenden Listing 1 ist das ganze Programm abgedruckt.

Zwei Dinge fallen dabei auf:
- Am rechten Rand stehen Zahlen in eckigen Klammern. Es sind Prüfzahlen, die dann entstehen, wenn das Programm mit der Eingabehilfe »Checksummer 64 V3« eingegeben wird, welche Sie in diesem Sonderheft auf Seite 159 finden.
- Viele Zeilen sind bisher in der Beschreibung nicht vorgekommen, zum Beispiel die Zeilen 40, 50.
  Sie dienen der besseren Lesbarkeit des Listings.

12 Lesbarkeit und Struktur eines Programms

Eine Methode, das Listing lesbar anzulegen, besteht darin, Leerzeichen an entsprechenden Stellen einzugeben. Basic macht nämlich keinen Unterschied zwischen:

PRINT A$; B$ und PRINTA$;B$.

Die erste Version braucht zwar drei Speicherplätze mehr – eben die drei Leerzeichen, aber Version 2 ist schlecht lesbar.

Zur besseren Lesbarkeit gehört auch, daß ab und zu eine Leerzeile eingefügt wird, die den Programmablauf nicht beeinflußt. Im Programm sind dies die Zeilen 30, 270 und 290. Man erreicht dies durch einen Doppelpunkt nach der Zeilennummer, ohne Befehl.

Eine dritte Art, das Listing zu verbessern, ist der Befehl

REM

Er kommt von Remark und bedeutet »Bemerkung«. Mit diesem Befehl kann man Text in das Listing eingeben, der aber vom Programm völlig ignoriert wird. Im Beispiel sind dies die Zeilen 40 und 380.

Basic-Befehl Nr. 14 REM
Dieser Befehl erlaubt, einen Text oder Zeichen – mit Zeilennummer versehen – in den Ausdruck eines Programms (Listing) einzufügen, ohne daß er im Ablauf des Programms erscheint. Es werden dabei keine Anführungszeichen verwendet.
Dieses Verfahren erhöht die Lesbarkeit des Listings.

Der vierte Punkt, der Ihnen im Listing 1 auffallen müßte, ist die Tatsache, daß bei allen PRINT-Befehlen der Text auf der linken Seite zwar mit einem Anführungszeichen anfängt, aber rechts nicht mit dem zweiten Anführungszeichen aufhört.

Das ist am Ende einer Zeile erlaubt, weil ja die Zeile mit der RETURN-Taste abgeschlossen worden ist.

Der fünfte Punkt betrifft die einzelnen Blöcke des Programms. In Listing 1 – und natürlich schon bei der Planung des Programms – habe ich einzelne Gruppen gebildet, die dem Programm eine logische und übersichtliche Struktur geben.

Diese Methode nennt man »Strukturiertes Programmieren«, was besonders durch die sogenannten »Unterprogramme« ermöglicht wird. Diese Unterprogramme kommen erst in Lektion 24 an die Reihe. Fürs erste genügt ein Hinweis darauf. Was halten Sie übrigens von einer kleinen Pause? Verdient haben Sie sich diese bestimmt.

13 Programmierte Steuertasten und der ASCII-Code

In Zeile 10 des Programms »Zahlen Raten« kommt ein invertiertes Zeichen in Anführungszeichen vor. Es ist das Symbol der Steuertaste CLR, eingebettet in ein Programm. Was bedeutet das?

Der Computer nützt hier eine Eigenschaft des PRINT-Befehls aus. PRINT druckt bekanntlich sowohl im Direkt- als auch im Programm-Modus alle Zeichen auf den Bildschirm, die in Anführungszeichen hinter ihm stehen.

→ PRINT"AAABBB"

druckt nach RETURN diese sechs Buchstaben aus. Was ist aber mit den Steuertasten CLR, HOME INST, DEL CRSR rauf/runter/links/rechts, und was ist mit den Farbtasten?

Merken wir uns:
1. Leerzeilen, die nur im Listing auftreten, das Programm aber nicht beeinflussen, erzeugt man durch einen Doppelpunkt hinter der Zeilennummer
2. Bei der Abarbeitung einer Programmzeile ignoriert Basic alle Leerzeichen, solange sie nicht zwischen Anführungszeichen stehend zu einem String gehören. Sie können daher weggelassen werden.
3. Am Ende einer Zeile kann das abschließende Anführungszeichen weggelassen werden.

Versuchen Sie es! Sie werden sehen, daß auch diese Tasten, zwischen Anführungszeichen, ein Symbol auf dem Bildschirm erzeugen, und zwar immer invertiert (also mit vertauschten Farbwerten für Vorder- und Hintergrund). In einem PRINT-Befehl erscheinen diese invertierten Zeichen nur im Listing.

Bei der Ausführung des PRINT-Befehls wird ihre Funktion ausgeführt !!

Das müssen Sie ausprobieren. Geben Sie einen PRINT-Befehl mit Buchstaben, gefolgt von der CTRL-Taste, gleichzeitig gedrückt mit der 8-Taste (YEL) und dann wieder Buchstaben ein.

```
10 PRINT"{CLR}                                  <186>
20 PRINT"******** ZAHLEN RATEN *******          <180>
30 PRINT                                        <132>
40 :                                            <016>
50 REM+++++++ VORBEREITUNG ++++++++++++         <244>
60 :                                            <036>
70 Z=INT(RND(0)*100)+1                          <060>
80 V=0                                          <065>
90 :                                            <066>
100 REM+++++++ RATEN +++++++++++++++++          <082>
110 :                                           <086>
120 PRINT"ICH HABE MIR EINE ZAHL                <144>
130 PRINT"ZWISCHEN 1 UND 100 GEMERKT.           <252>
140 PRINT                                       <242>
150 INPUT"WELCHE ZAHL IST ES ";R                <020>
160 V=V+1                                       <148>
170 :                                           <146>
180 REM+++++++ PRUEFEN +++++++++++++++          <096>
190 :                                           <166>
200 IF R>Z THEN PRINT R;:GOTO 230               <052>
210 IF R<Z THEN PRINT R;:GOTO 240               <126>
220 IF R=Z THEN PRINT R;:GOTO 250               <152>
230 PRINT"IST ZU GROSS":GOTO 140                <195>
240 PRINT"IST ZU KLEIN":GOTO 140                <043>
250 PRINT"IST RICHTIG"                          <067>
260 PRINT"SIE HABEN";V;"VERSUCHE ";             <020>
270 PRINT"BENOETIGT                             <030>
280 :                                           <002>
290 REM+++++ WIEDERHOLUNG ODER ENDE ++++        <153>
300 :                                           <022>
310 PRINT:PRINT:PRINT                           <181>
320 PRINT"NOCH EINMAL (J/N) ?                   <040>
330 GET A$: IF A$=""THEN 330                    <205>
340 IF A$="J"THEN 10                            <137>
350 IF A$<>"N" THEN 320                         <191>
360 PRINT:PRINT:PRINT"AUF WIEDERSEHEN           <017>
370 END                                         <118>
```

Listing 1. Das erste Programm – einfaches Zahlenraten

→ PRINT"AAA {CTRL 8} BBB"

Statt (CTRL 8) sehen Sie auf dem Bildschirm ein invertiertes Pi-Symbol.

Sobald Sie jetzt <RETURN> drücken, werden die B und alles folgende in Gelb gedruckt. Wenn Sie die ursprüngliche Farbe wiederherstellen wollen, drücken Sie entweder auf die CBM-Taste und die 7 gleichzeitig, oder aber auf STOP und RESTORE.



Genauso wie mit den Farb-Tasten geht es mit den Cursortasten. Zur Abwechslung machen wir das im Programm-Modus.

```
→ 10 PRINT"ZZZ {4mal CRSR-rechts} XXX"
```

Nach RUN werden die Z noch normal, die X aber um vier Stellen nach rechts versetzt gedruckt.

Die folgende Tabelle zeigt Ihnen die invertierten Steuerzeichen zur leichteren Identfizierung mit der Funktionsbeschreibung. An einem Beispiel wollen wir das verdeutlichen:

Funktion	Taste(n)	Zeichen	ASCII
schwarz	<CTRL 1>		144
weiß	<CTRL 2>		5
rot	<CTRL 3>		28
lila	<CTRL 4>		159
purpur	<CTRL 5>		156
grün	<CTRL 6>		30
blau	<CTRL 7>		31
gelb	<CTRL 8>		158
orange	<CBM 1>		129
braun	<CBM 2>		149
hellrot	<CBM 3>		150
hellgrau	<CBM 4>		151
mittelgrau	<CBM 5>		152
hellgrün	<CBM 6>		153
hellblau	<CBM 7>		154
dunkelgrau	<CBM 8>		155
Revers ein	<CTRL 9>		18
Revers aus	<CTRL 0>		146
Cursor rauf	<SHIFT CRSR>		145
Cursor ab	<CRSR>		17
Cursor links	<SHIFT CRSR>		157
Cursor rechts	<CRSR>		29
Schirm löschen	<SHIFT CLR/HOME>		147
Cursor Home	<CLR/HOME>		19
Klein-/Groß-buchstaben	<SHIFT CBM>		14
Großbuch-staben/Zeichen	<SHIFT CBM>		142
Funktionstaste F1	<F1>		133
F2	<SHIFT F1>		137
F3	<F3>		134
F4	<SHIFT F3>		138
F5	<F5>		135
F6	<SHIFT F6>		139
F7	<F7>		136
F8	<SHIFT F7>		140
Insert	<SHIFT INST/DEL>		148
Delete	-		20
Return	-		13
Shift-Return	-		141
Lock (Sperre der Zeichensatzumschaltung)	-		8
Unlock (Sperre aufheben)	-		9
1. Zeichensatz	-		142
2. Zeichensatz	-		14

Wenn Sie innerhalb einer Programmzeile nach einem Anführungszeichen ein invertiertes <]> finden, dann führt das Programm an dieser Stelle mit dem Cursor eine Bewegung durch: Es wird um eine Position nach rechts bewegt. Der ASCII-Code wird in Lektion 14 erläutert.

Besonders zu beachten sind dabei die letzten acht Steuerfunktionen. Sie verfügen zwar über über spezifische Steuerzeichen, können aber nicht mit Tasten direkt erzeugt werden.

Ihre Funktion kann nur dadurch erzielt werden, daß beim Eintippen für sie zuerst eine Leerstelle freigelassen wird. Nach <RETURN> muß man dann mit dem Cursor auf diese Leerstelle fahren, dann REVERS-ON, also <CTRL 9> drücken und anschließend das in der Tabelle dargestellte (reverse) Zeichen eingeben. Diese Aktion wird wieder mit <RETURN> abgeschlossen.

Ganz rechts in der Tabelle der Steuertasten ist eine Spalte von Zahlen, mit der Überschrift ASCII. Was dahintersteckt, sei kurz beschrieben.

Alle Computer verwenden intern irgendwelche Code-Zahlen, um die Zeichen, Buchstaben und Zahlen im Rechenwerk, im Speicher und in den anderen Einheiten des Computers darzustellen.

Theoretisch kann das jeder Computerhersteller machen, wie er will. Nur, wenn Daten von einem Gerät an ein anderes geliefert werden, zum Beispiel vom Computer an einen Drucker, müssen die Daten einem international standardisierten Code entsprechen. Dieser Standard heißt »American Standard Code for Information Interchange«, abgekürzt ASCII (Aussprache: »aski«).

Jedes Zeichen, jede Zahl und jede Funktion hat seinen eigenen Code-Wert. Sie sind (fast) alle in einer Tabelle im Anhang F des Commodore-Handbuches aufgelistet. In der Tabelle sind ganz einfach diese Werte nur für die Steuertasten ausgewählt und dargestellt.

Wir können aber ganz leicht ein kleines Programm schreiben, das uns die Abfrage aller ASCII-Codes gestattet. Dazu muß ich allerdings einen Befehl vorwegnehmen, den ich in der nächsten Lektion erkläre. Aber das macht nichts – Sie können mir sicher folgen.

```
→ 10 INPUT A$
  20 A=ASC(A$)
  30 PRINT A
  40 GOTO 10
```

Der Pfiff liegt natürlich im neuen Befehl ASC(A$) in Zeile 20. Er wandelt den ersten Buchstaben des per INPUT eingegebenen und mit <RETURN> abgeschlossenen Strings in seinen ASCII-Codewert um, den wir in Zeile 30 ausdrucken.

Zeile 40 verschafft dem Programm die Wiederholbarkeit, die bei INPUT nur durch die STOP- und RESTORE-Taste aufgehoben werden kann. Jetzt können Sie nach Herzenslust alle Tasten und Tastenkombinationen nach ihrem ASCII-Code abfragen und so die unvollständige Liste im Commodore-Handbuch vervollständigen.

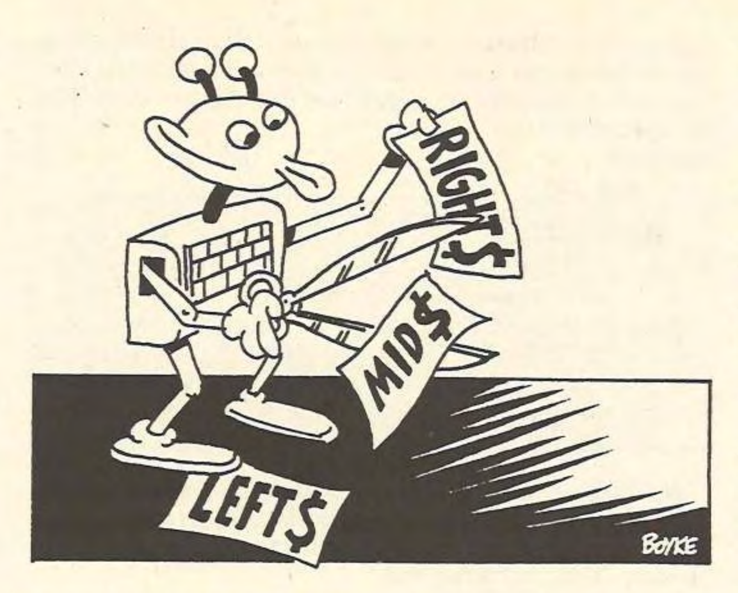
14 String-Befehle

Strings – auf deutsch »Zeichenketten« – haben wir ausführlich in Lektion 3 und Lektion 6 behandelt. Basic kennt mehrere Befehle, mit denen Strings verändert, verglichen und sonst irgendwie manipuliert werden können.

14.1. Strings und der ASCII-Code

Da mit dem PRINT-Befehl ja auch Daten an ein anderes Gerät gegeben werden, nämlich an den Bildschirm, versteht er die ASCII-Werte auch.

Der Tabelle im Handbuch entnehmen wir zum Beispiel den ASCII-Wert für das A, er ist 65. Um mit dieser Zahl den



Buchstaben A auf den Bildschirm zu zaubern, gibt es einen weiteren Basic-Befehl:

CHR$(ASCII-Wert)

Er ist eine Abkürzung von »Character«, was auf deutsch »Zeichen« bedeutet. Das String-Symbol muß immer dahinter stehen.

Basic-Befehl Nr. 15 CHR$(X)
- wandelt die ASCII-Codezahl X in ihr entsprechendes Zeichen um
- die Zuordnung der ASCII-Codes und der Zeichen ist im Handbuch als Anhang enthalten

In Verbindung mit dem PRINT-Befehl sieht der CHR$-Befehl so aus:

```
→ PRINT CHR$(65)
```

Diese Zeile, sowohl im Direkt-Modus als auch in einer Programmzeile, bewirkt dasselbe wie PRINT"A". Sie druckt den Buchstaben A auf den Bildschirm.

Was bringt denn das, werden Sie jetzt fragen. Die gute alte Anführungszeichen-Methode ist doch viel praktischer und kürzer. Aber die Methode der ASCII-Codewerte kann dennoch recht vielseitig sein.

Mit Zahlen, auch mit Codezahlen, kann man nämlich rechnen. Geben Sie bitte folgende Zeilen ein:

```
→ 10 X=64
  20 X=X+1
  30 PRINT CHR$(X);
  40 IF X=90 THEN END
  50 GOTO 20
```

In Zeile 10 geben wir der numerischen Variablen X den Wert 64, das ist um 1 weniger als die Codezahl des Buchstaben A.

In Zeile 20 wird eine Zählschleife begonnen mit der Erhöhung von X um 1.

Zeile 30 druckt das dieser Zahl entsprechende Zeichen aus. Im ersten Durchlauf ist es das A.

Zeile 50 schließt die Zählschleife durch den Rücksprung auf Zeile 20. Dadurch werden alle Zeichen vom Codewert 65 bis 90 - das ist das Alphabet von A bis Z - ausgedruckt.

Die Prüfzeile 40 wartet, bis das Z den Wert 90 erreicht und beendet dann das Programm. Nur mit Buchstaben in Anführungszeichen wäre dieses Programm recht lang geworden.

Der Befehl CHR$ wird hauptsächlich bei der Verarbeitung von Strings, also bei Erkennen, Vergleichen und Verändern von Wörtern eingesetzt. Er gehört zu einer ganzen Gruppe von Befehlen, die alle mit der String-Verarbeitung zu tun haben. Wir werden sie noch einzeln kennenlernen.

Der direkte Nachbar zu CHR$ ist der Befehl

ASC (String)

Er ist die exakte Umkehrung des CHR$-Befehls. ASC ist die Abkürzung von ASCII.

Erinnern Sie sich, ich habe diesen Befehl schon kurz verwandt, in Lektion 13 bei der Vorführung des ASCII-Codes.

Der Befehl wandelt das erste Zeichen des in Klammern stehenden Strings in seinen ASCII-Codewert um.

```
→ PRINT ASC("WORT")
```

Diese Zeile druckt den ASCII-Wert von W aus, es ist 87.

Basic-Befehl Nr. 16 ASC(String)
wandelt das erste Zeichen des Strings in seinen ASCII-Codewert um

Als Beispiel verwende ich die schon mehrfach geübte Frage nach Wiederholung oder Ende eines Programms, diesmal aber durch Vergleich der ASCII-Werte.

```
→ 10 PRINT"PROGRAMM-ANFANG"
  .
  .      (Programm)
  .
  100 PRINT"NOCH EINMAL (J/N) ?"
  110 GET A$:IF A$="" THEN 110
  120 X=ASC(A$)
  130 IF X=74 THEN 10
  140 IF X <> 78 THEN 100
  150 END
```


In Zeile 10 soll das Programm beginnen - welches, interessiert uns hier nicht weiter.

Am Ende dieses Programms fordert Zeile 100 zur Eingabe von JA oder NEIN auf.

Zeile 110 enthält das gewohnte Bild der GET-Schleife, die wartet, bis ein Zeichen im Tastaturpuffer erscheint.

Zeile 120 wandelt das erste Zeichen des eingegebenen Wortes in seinen ASCII-Code um. Aus der ASCII-Codetabelle wissen wir, daß dem erwarteten J die Zahl 74, dem N aber die Zahl 78 entspricht. Alle anderen Werte sollen zurückgewiesen werden.

Ist X=74, dann springt Zeile 130 an den Anfang des Programms zurück.

Ist X nicht 78, wird durch Zeile 140 die Eingabe wiederholt.

Im Fall, daß X=78, fällt die Prüfung in Zeile 140 durch, und das Programm macht in Zeile 150 Schluß.

Auch dieses Programm ist länger als die alte Methode, aber es zeigt wenigstens die Arbeitsweise von ASC.

14.2. Das Zerteilen von Strings

Der Befehl ASC hat den Nachteil, daß er immer nur das erste Zeichen des Strings zur Umwandlung nimmt. Um ein ganzes Wort in seine einzelnen ASCII-Codewerte zu zerlegen, brauchen wir eine Methode, welche uns gestattet, einzelne Zeichen des Wortes abzuschneiden bzw. herauszupicken. Ich müßte eigentlich String statt Wort sagen, denn das alles gilt natürlich auch für Zeichen und Symbole.

Es wäre schön, wenn man dieses Abschneiden am Anfang, in der Mitte oder am Ende dieses Wortes bzw. Strings machen könnte.

Basic bietet uns erfreulicherweise alle drei Möglichkeiten:
- LEFT$ schneidet vom linken Rand des Strings Zeichen heraus
- RIGHT$ tut dasselbe auf der rechten Seite
- MID$ pickt Teile aus der Mitte des Strings heraus

Natürlich bieten diese Befehle auch die Möglichkeit, anzugeben, wie viele Zeichen abgetrennt werden sollen.

Bei LEFT$ und RIGHT$ ist das nur eine einzige Zahlenangabe.

Bei MID$ brauchen wir zwei Zahlen. Die eine gibt an, ab wo herausgepickt werden soll, die zweite sagt wieviel.

Im Beispiel wird das schnell klar:

```
10 A$= "MOTORHAUBENVERSCHLUSS"
20 B$ = LEFT$(A$,5)
30 PRINT B$
40 C$ = RIGHT$(A$,7)
50 PRINT C$
60 D$ = MID$(A$,6,5)
70 PRINT D$
```

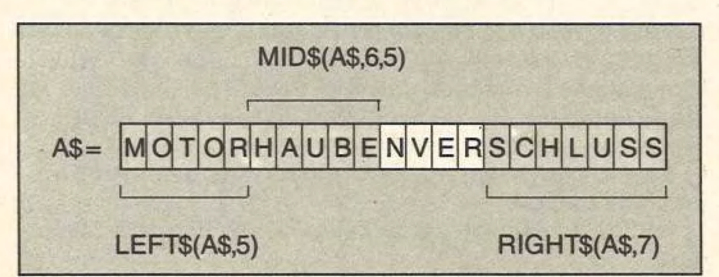
Zeile 10 legt uns die Basis mit der Zuweisung des langen Wortes an die String-Variable A$. Jetzt geht's ans Beschneiden. Mit Zeile 20 fangen wir an der linken Seite des Wortes an. Hinter dem Befehl LEFT$ steht in der Klammer zuerst der String, der zerpflückt werden soll, nämlich A$. Die zweite Angabe - nach dem Komma - gibt an, wie viele Zeichen abgeschnitten werden sollen, im Beispiel sind es fünf. Diese fünf Zeichen ergeben gerade den neuen String »MOTOR«, der in Zeile 30 ausgedruckt wird.

Die Schreibweise und Funktion von RIGHT$ ist identisch, halt nur auf der rechten Seite des Wortes A$ wirkend. In Zeile 40 erhält dieser Teilstring den Variablennamen C$. Zeile 50 druckt also die rechten sieben Zeichen des Strings A$ aus, was das Wort »SCHLUSS« ergibt.

Interessant wird es in Zeile 60 beim Befehl MID$. Nach der Angabe des betroffenen Strings A$ steht zuerst die Nummer des Zeichens, ab dem von links gezählt herausgeschnippelt werden soll, die zweite Zahl daneben gibt an, wie viele Zeichen es sein sollen.

In unserem Beispiel ist das sechste Zeichen von links das »H«, ab da fünf Zeichen weiter - inklusive des »H« - ergeben das Wort »HAUBE«, das in Zeile 70 ausgedruckt wird.

Das folgende Bild soll das verdeutlichen:



Ich empfehle Ihnen, ein bißchen zu experimentieren, am besten mit nur einer Zeile

```
100 PRINT LEFT$("DRACHEN",2)
RUN 100
```

Erstaunt Sie die Schreibweise? Das sollte jetzt eigentlich nicht mehr passieren.

Der einzige Unterschied zu vorher ist, daß wir vorher den String zuerst einer String-Variablen zugeordnet haben, wodurch er im Speicher aufbewahrt wird. Hier schreiben wir den String direkt an die Stelle der Variablen, der Computer merkt ihn sich halt nicht. Aber mit RUN 100 können Sie leicht den Befehl immer wieder ausführen, nachdem Sie mit dem Cursor auf die Zahl gefahren sind und diese verändert haben.

Genauso geht es mit dem Befehl RIGHT$, weswegen ich ihn hier auslasse.

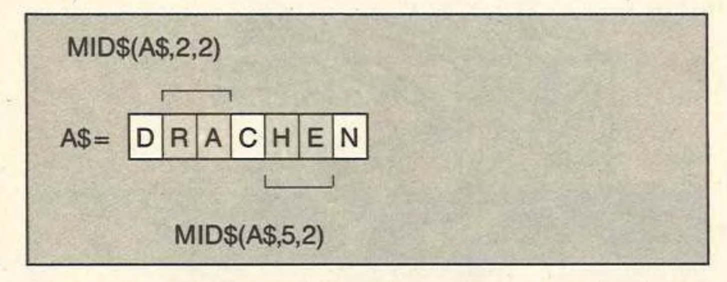
MID$ ist der kniffligste der drei Befehle.

```
200 PRINT MID$("DRACHEN",2,5)
```

ergibt »RACHE«. Wenn Sie die zweite Zahl ganz weglassen, erhalten Sie »RACHEN«.

Die Frage stellt sich, ob wir aus dem Wort »DRACHEN« mit diesem Befehl zwei getrennte Teile herausholen und so zusammensetzen können, daß zum Beispiel das Wort »RAHE« entsteht.

Das geht!!



Wir müssen bloß zuerst den Teil »RA« und in einem zweiten Schritt den Teil »HE« herauszwicken und dann beide Teilstrings zu einem neuen String zusammenfügen. Die folgenden Zeilen schaffen das:

```
10 A$="DRACHEN"
20 X$=MID$(A$,2,2)
30 Y$=MID$(A$,5,2)
40 Z$=X$+Y$
50 PRINT Z$
```

Ich glaube, das brauche ich nicht zeilenweise zu erläutern.

Jetzt soll aber noch ein bißchen Mathematik dazukommen. Es ist ja nicht zwingend vorgeschrieben, daß die Zahlen in der Klammer hinter dem String-Befehl konstant sind. Schauen Sie sich das folgende Beispiel an:

```
10 A$="DRACHEN"
20 X=X+1
30 B$=LEFT$(A$,X)
40 PRINT B$
50 IF X=7 THEN END
60 GOTO 20
```

Dieses Programm druckt uns das folgende Muster aus:

D
DR
DRA
DRAC
DRACH
DRACHE
DRACHEN

Das Geheimnis liegt in Zeile 20, in der die Zahl der abzuschneidenden Zeichen nicht konstant, sondern durch die Zählschleife mit X von 1 bis 7 hochgezählt wird.

Und jetzt kommen wir unserem früheren Wunsch, alle Buchstaben eines Wortes in ihren ASCII-Code umzuwandeln, schon näher.

Erinnern Sie sich, diese Umwandlung liefert uns der ASC-Befehl, aber immer nur für den ersten Buchstaben eines Wortes. Alle weiteren Buchstaben werden von diesem Befehl nicht berücksichtigt.

Basic-Befehl Nr. 17 LEFT$(X$,A)
- schneidet vom String X$ von links her A-Zeichen ab und bildet daraus einen neuen String

Nr. 18 RIGHT$(X$,A)
- macht genau dasselbe wie Nr. 17, aber von rechts her

Nr. 19 MID$(X$,B,A)
- schneidet vom String X$ von links her ab dem B-Zeichen insgesamt A-Zeichen heraus
- bei MID$ kann die zweite Zahl »A« weggelassen werden. Dann schneidet es ab dem B-Zeichen den Rest des Strings ab

Was wir brauchen, ist so ein Muster:
DRACHEN
RACHEN
ACHEN
CHEN
HEN
EN
N

Mit LEFT$ geht das nicht, denn wir schneiden ja an der rechten Seite ab. Außerdem fangen wir mit der vollen Wortlänge von 7 an und reduzieren sie laufend.

Also, der RIGHT$-Befehl muß her, kombiniert mit einer Zählschleife, die aber rückwärts zählt:

```
→ 10 A$="DRACHEN"
  20 X=7
  30 B$=RIGHT$(A$,X)
  40 PRINT B$
  50 X=X-1
  60 IF X=0 THEN END
  70 GOTO 30
```

Die rückwärts zählende Schleife wird durch die Festlegung des Anfangswertes in Zeile 20 und durch das laufende Vermindern der Zählvariablen in Zeile 50 gebildet.

Um noch den ASCII-Codewert der jeweils ersten Buchstaben zu erhalten, müssen wir nur Zeile 40 erweitern zu:

```
→ 40 PRINT B$ "=" ASC(B$)
```

Nach dem bisherigen Ausdruck des Teilstrings B$ folgt das Gleichheitszeichen – als String zwischen Anführungszeichen gesetzt – und danach der umgewandelte Wert des ersten Buchstabens von B$.

Ich muß Ihnen gestehen, daß das alles nur zur Übung war. Denn am elegantesten geht es mit dem Befehl MID$.

Wir kehren wieder zur hochzählenden Schleife zurück und zwicken von links her der Reihe nach die Buchstaben heraus, immer nur einen, mit dem Befehl MID$(A$,X,1)

```
→ 10 A$="DRACHEN"
  20 X=X+1
  30 B$=MID$(A$,X,1)
  40 PRINT B$ "=" ASC(B$)
  50 IF X=7 THEN END
  60 GOTO 20
```

Es gibt noch einen STRING-Befehl, der uns das Zählen der Zeichenzahl für die Prüfung in Zeile 50 abnimmt. Er heißt

LEN(String)

Er ist eine Abkürzung von Length, was auf deutsch »Länge« heißt. Er wird seinem Namen gerecht und bestimmt die Länge des Strings.

Wenn Sie direkt eingeben:

```
→ PRINT LEN("DRACHEN")
```

dann erhalten Sie die Zahl 7 als Resultat.

Um ihn im obigen letzten Programmbeispiel einzusetzen, brauchen wir lediglich die Zeile 50 ändern:

```
→ 50 IF X=LEN(A$) THEN END
```

So prüft sie X so lange, bis es den Wert von LEN(A$), nämlich 7, erreicht hat.

Basic-Befehl Nr. 20 LEN(String)
- gibt die gesamte Länge des Strings einschließlich der Leerzeichen und Steuerzeichen als Zahl aus
- in der Klammer kann ein String oder aber eine String-Variable sein, der ein String zugewiesen worden ist

14.3. Die Addition von Strings

In Abschnitt 6.2, bei der Diskussion der String-Variablen, haben wir bereits mit zusammengehängten Strings experimentiert, unter Verwendung des PRINT-Befehls:

```
110 A$="HOLZ"
115 B$="FEUER"
120 PRINT A$B$
```

Das Resultat der Zeile 120 ist das Wort HOLZFEUER. Dabei ist es egal, ob Sie in Zeile 120 zwischen den beiden Stringvariablen ein Semikolon, einen Zwischenraum oder, wie oben, beide aneinanderschreiben.

Wenn wir aber dem »HOLZFEUER« eine eigene Variable zuordnen wollen, müssen wir A$ und B$ addieren, unter Verwendung des »+«-Zeichens.

```
125 C$=A$+B$
130 PRINT C$
```

Zeile 120 und Zeile 130 ergeben das gleiche Resultat, nur hat sich der Computer die neue Stringvariable C$ gemerkt; das Resultat der Addition ist dadurch auch noch später verfügbar.

Nicht ganz so selbstverständlich wie Strings mit Buchstaben sind Strings, die aus Zahlen bestehen.

```
135 X$="7"
140 Y$="5"
145 Z$=X$+Y$
150 PRINT Z$
```

Zeile 150 druckt die Zahl 75 aus. Bemerkenswert ist dabei, daß in der String-Form die Ziffern ohne den sonst üblichen freien Platz für ein eventuelles negatives Vorzeichen gedruckt werden.

Wir haben in Lektion 13 gelernt, Steuerzeichen wie zum Beispiel »CURSOR-LINKS«, »REVERSE-ON« innerhalb von Anführungszeichen in unsere Programme einzubauen. Nun, da diese auch Strings darstellen, können wir sie ebenfalls mit anderen Strings zusammensetzen.

```
155 L$=" {CTRL+RVS-ON}"
160 M$=" {CTRL+RVS-OFF)}"
165 PRINT L$+B$+M$+A$
```

Zeile 165 druckt das Wort FEUERHOLZ aus, wobei der Wortteil FEUER in reverser Darstellung erscheint. Zuguterletzt sollen die Zeilen 170 bis 195 zeigen, daß auch die Grafikzeichen, die vorn auf den Tasten stehen und auf die mit der SHIFT- oder der CBM-Taste umgeschaltet wird, als Strings addiert werden können.

```
170 Q$=" {SHIFT+O/CBM+Y/SHIFT+P} "
175 R$=" {3mal CRSR-LINKS} "
180 S$=" {CRSR-DOWN} "
185 T$=" {SHIFT+L/CBM+P/SHIFT+@} "
190 U$=Q$+R$+S$+T$
195 PRINT U$
```

In Zeile 190 wird eine Stringvariable U$ definiert, die aus sechs Zeichen und mehreren Cursorbewegungen besteht, wodurch ein Rechteck gezeichnet wird. In Zeile 195 wird dieses Rechteck ausgedruckt.

Wenn wir die Zeile 195 so abändern:

```
195 PRINT U$;:GOTO 195
```

wird in einer »ewigen« Schleife das Rechteck diagonal über den Bildschirm gedruckt.

Merken wir uns:
1. Durch Addition von einzelnen Strings werden neue, längere Strings gebildet.
2. Aus einzelnen Buchstaben pro Variable entstehen so einzelne Wörter pro Variable, aus diesen wiederum ein ganzer Satz pro Variable.
3. Eine String-Variable kann maximal 255 Zeichen enthalten.

Eine derart zusammengesetzte String-Variable kann aus Buchstaben, grafischen Zeichen, Ziffern und Steuerzeichen bestehen.



Die aus einzelnen Zeichen zusammengesetzte Form wird wie ein einzelnes neues Zeichen behandelt!

Diese Additionsmethode für Strings kann in vielen Programmarten sehr sinnvoll eingesetzt werden.

14.4. Der Größenvergleich von Strings

Beim Stringvergleich mit > und < muß der Programmierer wissen, was einen »größeren« String ausmacht. Ich möchte es Ihnen mit einem kleinen Versuch deutlich machen.

```
310 INPUT "STRING #1";A$
320 IF A$="@" THEN END
330 INPUT "STRING #2";B$
```

Wir geben also zwei Strings in den Zeilen 310 und 330 ein. Zeile 320 erlaubt uns den Aussprung aus der Schleife mit der Klammeraffen-Taste.

Jetzt vergleichen wir die beiden Strings:

```
340 IF A$ > B$ THEN PRINT "#1"
350 IF A$ < B$ THEN PRINT "#2"
360 GOTO 310
```

Die Vergleiche in Zeile 340 und 350 wählen beide den größeren String zum Ausdrucken aus.

Lassen Sie das Programm laufen und experimentieren Sie ein bißchen.

Sie werden zum Beispiel sehen:
»A« ist größer als »B«
»ACB« ist größer als »ABC«
»TISCHE« ist größer als »TISCH«
»WORT 2« ist größer als »WORT 1«
Ich will Ihnen verraten, was der Computer macht.

Er vergleicht die ASCII-Codewerte der einzelnen Buchstaben von links aus. Zur Erinnerung, der ASCII-Code ist eine Zahl, mit der der Computer ein Zeichen intern kennzeichnet (siehe Lektion 13).

Das A hat den ASCII-Wert 65, B den Wert 66, C den Wert 67. Sowohl bei ACB als auch bei ABC ist das erste Zeichen gleich, aber von den zweiten Zeichen ist C größer als B. Im Beispiel TISCHE-TISCH macht das zusätzliche E den Unterschied im String 2.

Beim Vergleich von WORT 2 mit WORT 1 hat die Ziffer 2 einen höheren ASCII-Codewert als die Ziffer 1.

Das Beispiel oben zeigt, daß natürlich auch Zahlen-Strings größer oder kleiner sein können, genauso wie gemischte Strings (WORT 2).

Merken wir uns:
1. Werden Strings miteinander verglichen, vergleicht der Computer schrittweise die ASCII-Codewerte der einzelnen Zeichen. Die erste Ungleichheit entscheidet, wobei dann der String mit dem höheren ASCII-Codewert »größer« ist.
2. Da der Stringvergleich sich auf ASCII-Codewerte bezieht, wird kein Unterschied gemacht zwischen Buchstaben, Zeichen oder Ziffern.

14.5. Verwandlung von Strings und Zahlen

Im Abschnitt 14.3 haben wir gesehen, daß Zahlen als Strings verarbeitet werden, wenn sie in Anführungszeichen stehen. Das hat unter anderem den Vorteil, daß kein Leerzeichen für das Vorzeichen reserviert wird.

```
10 A$="123"
15 PRINT A$
```

druckt die Zahl ganz an den linken Rand.

Es gibt zwei Befehle, die uns erlauben, Strings in Zahlen und Zahlen in Strings zu verwandeln:

Der eine der beiden heißt VAL(A$). Sein Name ist von VALUE (Wert) abgeleitet. Er wandelt A$ in einen Zahlenwert um, falls überhaupt in A$ eine Zahl vorkommt.

```
20 PRINT VAL(A$)
```

Diese Zeile druckt ebenfalls die Zahl 123 aus, aber eben als Zahl, das heißt mit einer Leerstelle vor ihr.

Ist in dem String keine Zahl enthalten, dann wird der Wert Null ausgegeben.

```
25 B$="ABC"
30 PRINT VAL(B$)
```

Wenn wir statt »ABC« den String »A2C« schreiben, erhalten wir immer noch Null, weil der String mit einem Buchstaben anfängt.

Ist B$ aber »12C«, dann ergibt der VAL-Befehl die Zahl 12.

Basic-Befehl Nr. 21 VAL(A$)
- Der Befehl VAL(A$) liefert den numerischen Wert von A$
- Dieser String kann sowohl in dem $-Zeichen als auch in Anführungszeichen geschrieben sein
- VAL(A$) fragt den String A$ Zeichen für Zeichen nach einer Zahl ab. Ab dem ersten Zeichen, das keine Zahl ist, stoppt er die Suche und verwendet den Wert des bisherigen Teils weiter
- Wenn der String A$ keine Zahlen enthält, wird der Wert 0 gebildet

Die Umkehrung von VAL ist der Befehl STR$(X), abgeleitet von STRING. Er wandelt die Zahl X in einen String um. Wozu das gut ist, zeigen uns die nächsten Zeilen:

```
35 X=123
40 Y=456
45 PRINT X+Y
```

Das Resultat der Zeile 45 ist natürlich die Summe der beiden Zahlen, also 579, mit einer Leerstelle ausgedruckt.

```
50 PRINT X;Y
```

Diese Zeile dagegen setzt beide Zahlen nebeneinander, getrennt durch zwei (!) Leerstellen, eine für das Vorzeichen, die zweite für die Trennung zweier unabhängiger Zahlen:

```
 123  456
```

Diese zweite Trennung heben wir auf mit der Verwendung des STR$-Befehls:

```
55 PRINT STR$(X);STR$(Y)
```

Wir erhalten:

```
 123 456
```

Aber Sie sehen, daß X und Y immer noch Zahlen sind, obwohl sie über STR$ als Strings behandelt werden.

Erst die Zeile:

```
60 PRINT "123";"456"
```

macht echte Strings aus den Zahlen und vermeidet alle Zwischenräume.

```
123456
```

Leider können wir in dieser Version mit den Zahlen nicht mehr rechnen. Mit STR$ geht das aber, was die nächsten Zeilen 65 bis 75 zeigen.

```
65 FOR Z=0 TO 2
70 PRINT STR$(X+Z);STR$(Y)
75 NEXT Z
```

Basic-Befehl Nr. 22 STR$(X)
- der Befehl STR$(X) bildet aus dem Wert der Zahl X einen String
- dieser String behält die Leerstelle für das Vorzeichen

Wir erhalten die Zahlenreihen

```
 123 456
 124 456
 125 456
```

15 Ein Programm mit den String-Befehlen

Ich möchte mit Ihnen zusammen ein zweites Programm entwickeln, welches hauptsächlich die Stringbefehle verwendet und Ihnen ein bißchen Erfahrung in der String-Verarbeitung gibt.



Aufgabe:

Im ersten Teil des Programms sollen eingegebene Wörter in einen Geheimcode umgewandelt werden. Die verwandelten Wörter stehen alle auf dem Bildschirm.

Per Tastendruck soll dann der zweite Programmteil angesprungen werden, in dem die einzelnen Codewörter wieder in ihre ursprüngliche Form zurückgewandelt (decodiert) werden.

Der übersichtlichen schrittweisen Darstellung zuliebe verzichte ich auf ein Flußdiagramm. Wir »hacken« uns also durch.

Die Eingabe der Wörter machen wir zuerst über den GET-Befehl. Sie wissen, daß der GET-Befehl immer nur ein Zeichen annimmt, aber keine ganzen Wörter. Wir müssen also erst eine Schleife bauen, um die einzelnen Zeichen zu einem Wort zusammenzusetzen.

```
100 GET W$: IF W$="" THEN 100
120 PRINT W$;
130 A$=A$+W$
150 GOTO 100
```

Zeile 100 ist die übliche Eingabe-Warteschleife mit GET.

Zeile 120 druckt jedes einzelne Zeichen W$ mit dem Semikolon aneinandergeklebt aus.

Zeile 130 ist etwas Neues. Es wird ein String mit Namen A$ dadurch gebildet, daß zu seinem jeweiligen Zeichen – am Anfang ist keines da – das gerade eingegebene Zeichen W$ dazugenommen wird. Auf diese Weise entsteht ein ganzes Wort A$.

Der Rücksprung in Zeile 150 läßt dieses Wort unendlich lange werden.

Wir brauchen also am Ende des Wortes einen Aussprung aus der Schleife. Ich schlage vor, das Wort mit der RETURN-Taste zu beenden. Deshalb müssen wir in Zeile 140 den ASCII-Code der RETURN-Taste abfragen. Er ist 13.

```
140 IF W$=CHR$(13) THEN 170
170 PRINT"**"
```

Die Abfrage kennen wir schon. Sie springt nach 170, wo vorläufig zum Ausprobieren zwei Sterne ausgedruckt werden.

Codier-/Decodier-Algorithmus

»Algorithmus« ist die Bezeichnung für einen logischen Vorgang oder eine Formel, in unserem Fall die Vorschrift, nach der das Wort A$ verschlüsselt werden soll.

Ich habe einen sehr leichten Algorithmus gewählt:

Der ASCII-Codewert jedes Zeichens wird um die Zahl seines Platzes im Wort erhöht, der erste Buchstaben um 1, der zweite Buchstaben um 2 und so weiter.

Das Decodieren geht entsprechend, indem immer die Platz-Ziffer abgezogen wird.

Also müssen wir zuerst das Wort A$ in seine ASCII-Codewerte umwandeln. Das haben wir schon einmal gemacht.

In Zeile 160 gebe ich ein Wort für A$ vor, aber nur zum Ausprobieren.

```
160 A$="WERT"
170 X=X+1
180 A=ASC(MID$(A$,X,1))
190 B=A+X
195 PRINT B
220 IF X <>  LEN(A$) THEN 170
```

A$ in Zeile 160 sei also das fertig eingegebene Wort.

Die Zeilen 170 und 220 bilden eine Zählschleife, deren Variable X von 1 bis zur Länge des eingegebenen Wortes A$ zählt und am Ende weiterspringt. In unserem Beispiel ist LEN(A$)=4 Buchstaben.

Zeile 180 kennen Sie schon. Sie wandelt jeden Buchstaben des Wortes A$ – einzeln wegen der letzten Ziffer 1 – von links ab dem X-Zeichen in seinen ASCII-Code um.

Zeile 190 ist die eigentliche Verschlüsselung oder Codierung. Hier wird eine Variable B dadurch gebildet, daß zum ASCII-Wert »A« des gerade umgewandelten Zeichens »A$« der jeweilige Wert von X dazugezählt wird.

Zeile 195 ist eingeschoben, um die neuen ASCII-Werte auszudrucken.

Starten Sie diesen Teil mit RUN 160, und Sie erhalten die neue Zahlenfolge 88, 71, 85 und 88.

Jetzt nehmen wir Zeile 160 heraus, oder besser noch, wir wandeln sie in eine REM-Zeile zur Schönschrift um.

Sie können jetzt mit RUN das Ganze laufen lassen, nämlich Eingabe eines Wortes bis zum Ausdruck der neuen Codezahlen. Jetzt allerdings erhalten Sie immer eine Zahl mehr, als Zeichen eingegeben wurden. Diese zusätzliche Zahl entsteht durch die RETURN-Taste.

Die nächsten drei Zeilen wandeln die verschlüsselten Codezahlen »B« in Buchstaben »B$« zurück und setzen sie zu einem verschlüsselten Wort »C$« zusammen.

```
→ 200 B$=CHR$(B)
  210 C$=C$+B$
  240 PRINT C$
```

Mit RUN erhalten Sie jetzt auch das verschlüsselte Wort ausgedruckt.

Um mehrere Wörter eingeben zu können, bilden wir mit dem GOTO-Befehl der Zeile 260 eine Schleife. Zeile 195 kann jetzt entfernt werden.

```
→ 195
  260 GOTO 100
```

Ein neuer Versuch klappt allerdings nicht.

Die Eingabe eines zweiten Wortes stößt auf Schwierigkeiten. Der Grund dafür liegt darin, daß das erste Wort A$ noch vorhanden ist. Wir müssen es einfach löschen. Und weil wir gerade beim Korrigieren sind: Auch die Zählvariable X muß vor der nächsten Eingabe auf 0 gesetzt werden. Der letzte Fehler besteht darin, daß in Zeile 220 der LEN-Befehl immer ein Zeichen mehr mißt, weil ja die RETURN-

Taste am Ende des eingegebenen Wortes für ihn auch als Zeichen gewertet wird.

Hier bietet sich zusätzlich die Gelegenheit, für die Schönschrift der Ausgabe etwas zu tun. Mir gefällt nämlich nicht, daß beim Ausdrucken zwischen den codierten Wörten kein Zwischenraum gelassen wird. Ich schiebe daher die Zeile 230 vor den PRINT-Befehl, um an den bestehenden String »C$« – das codierte Wort – ein Leerzeichen mit dem ASCII-Code 32 anzuhängen.

Die folgenden Zeilen korrigieren das alles.

```
→ 220 IF X <> LEN(A$)-1 THEN 170
  230 C$=C$+CHR$(32)
  250 A$=""
  260 X=0: GOTO 100
```

In Zeile 220 prüft der IF-THEN-Befehl auf ein Zeichen weniger, als das Wort plus RETURN-Taste lang ist. Zeile 250 zeigt, wie man eine Stringvariable durch Zuweisung eines Nullstrings löscht. Dasselbe tut Zeile 260 mit der numerischen Variablen X.

Jetzt läuft der erste Programmteil, der aus einem eingegebenen Wort ein verschlüsseltes Wort baut. Wir erhalten zum Beispiel für das Wort »WERT« den Code »XGUX«.

Das Decodieren stelle ich mir so vor, daß der Benutzer per Tastatur die Codes eingibt, welche er entschlüsseln will. Das wollen wir zur Übung diesmal mit INPUT machen.

```
→ 280 INPUT E$
```

Die nächsten Zeilen gleichen weitgehend den Zeilen 170 bis 260, da der Vorgang ja fast identisch ist. Eine Ausnahme ist die Zeile 190, da wir zum Decodieren ja die Zählvariable X abziehen müssen. Außerdem habe ich die beiden Zeilen 200 und 210 zu einem Ausdruck zusammengefaßt. Und schließlich braucht dem LEN-Befehl keine 1 abgezogen werden, da beim INPUT-Befehl im Gegensatz zum GET die eingegebene RETURN-Taste nicht zum Wort dazugerechnet wird. Also sieht das Decodieren so aus:

```
→ 280 INPUT E$
  290 X=X+1
  300 E=ASC(MID$(E$,X,1))
  310 D=E-X
  320 D$=D$+CHR$(D)
  330 IF X <> LEN(E$) THEN 290
  340 D$=D$+CHR$(32)
  350 PRINT D$
  360 E$=""
  370 X=0:GOTO 280
```

Mit RUN 280 können Sie diesen Programmteil separat testen. Jetzt müssen wir die beiden Teile noch zusammenbinden. Ich stelle mir vor, daß es praktisch ist, nach einigen eingegebenen Wörtern diese zu decodieren. Das Signal dazu soll die Funktionstaste <F1> sein. Die Arbeitsweise der Funktionstasten haben wir noch nicht durchgenommen. Springen Sie bitte auf die Lektion 16, bevor Sie mit diesem Programm weitermachen.

Wenn die Funktionstaste gedrückt wird, soll das Programm vom Codierteil auf den Decodierteil springen.

Diese Prüfung müssen wir ganz am Anfang des Codierteils vornehmen, und die GET-Eingabeschleife bietet sich dazu in hervorragender Weise an. Wir schieben also die folgenden drei Zeilen ein:

```
→ 110 IF W$=CHR$(133) THEN 270
  270 PRINT"DECODIEREN"
  285 IF E$="ENDE" THEN END
```

Zeile 110 verwendet den ASCII-Code von 133 der F1-Taste, um auf die Zeile vor dem INPUT-Befehl zu springen (270), wo ein Hinweis ausgedruckt wird.

Die Zeile 285 beendet auf simple Weise das Programm. Das einzige, was nicht passieren darf ist, daß das Wort »ENDE« als codierter Wert auftritt.

Merken wir uns:

1. Eine String-Variable kann dadurch »gelöscht« werden, daß man ihr einen Null-String zuweist mit dem Befehl A$="".
   Das entspricht dem Nullsetzen einer numerischen Variablen mit X=0
2. Funktionstasten und andere Steuertasten, die nicht auf dem Bildschirm erscheinen, können nur mit dem GET-Befehl abgefragt werden.
3. Ein String kann durch Hinzufügen von anderen Strings zu einem größeren String gleichen Namens erweitert werden: A$=A$+B$

Zur Bedienung des Programms ist zu sagen, daß Sie beim Decodieren lediglich die auf dem Bildschirm stehenden verschlüsselten Wörter einzeln ablesen und eintippen müssen.

Wem das zu mühsam ist, soll ein Programm für die automatische Decodierung schreiben !

In Listing 2 ist das komplette Programm wiedergegeben, auch hier mit den Prüfsummen der Eingabehilfe »Checksummer 64 V3«.

Und jetzt kommen im Nachtrag die Funktionstasten an die Reihe.

16 Die Funktionstasten

Ich möchte gern einen Rücksprung machen zu der Tabelle der Steuertaste und ihrer ASCII-Codewerte.

Da sehen Sie als untersten Block die Funktionstasten <F1> bis <F8> angegeben. In diesem Kurs werden sie da praktisch zum ersten Mal erwähnt. Jetzt wird es langsam Zeit, näher darauf einzugehen.

Wenn Sie eine der Funktionstasten drücken, passiert bekanntlich gar nichts – oder doch?

Nun, sie haben einen eigenen ASCII-Code. Mit einem kleinen Programm, das wir bei der Besprechung der ASCII-Codes verwendet haben, können wir diese Behauptung hier noch einmal überprüfen. Das Programm sah so aus:

```
→ 10 INPUT A$
  20 A=ASC(A$)
  30 PRINT A
  40 GOTO 10
```

Nicht nur die Funktionstasten, sondern auch alle anderen Steuertasten geben so ihren ASCII-Code preis.

Diesen Code können wir zur Abfrage der Tasten, ob sie nämlich gedrückt worden sind, verwenden.

Fügen Sie bitte die folgende Zeile 35 ein:

```
→ 35 IF A=65 THEN PRINT "SERVUS"
```

Das Programm funktioniert wie vorher, nur wenn wir das »A« drücken, dessen ASCII-Code 65 ist, wird der freundliche Gruß ausgedruckt. Statt 65 können Sie jeden beliebigen Codewert verwenden und damit das Drücken der entsprechenden Taste abfragen.

Halt! Das stimmt nicht ganz. Denn wenn Sie in Zeile 35 den Codewert einer der Funktionstasten eingeben, rührt sich kein SERVUS.

Warum nicht?

Vielleicht erinnern Sie sich, daß im Unterschied zum GET-Befehl, der im Tastaturpuffer nachschaut, ob dort ein Zeichen abgespeichert ist, der INPUT-Befehl alle eingegebenen Zeichen erst auf den Bildschirm bringt und sie von dort dann weiterverarbeitet.

Das ist des Rätsels Lösung, denn die Funktionstasten hinterlassen halt keine Spur auf dem Bildschirm, genauso wenig wie alle anderen Steuertasten. Das heißt, zu deren Abfrage müssen wir den GET-Befehl nehmen.

Ändern wir also die Zeile 10 ab:

```
→ 10 GET A$:IF A$="" THEN 10
```

Jetzt geht es!!

Zur Demonstration der Abfrage der Funktionstasten habe ich unser kleines Programm wie folgt erweitert:

```
→ 10 GET A$:IF A$="" THEN 10
  20 A=ASC(A$)
  30 PRINT A
  35 IF A=133 THEN PRINT CHR$(158)
  36 IF A=134 THEN PRINT CHR$(154)
  37 IF A=135 THEN PRINT CHR$(19)
  38 IF A=136 THEN PRINT CHR$(147)
  40 GOTO 10
```

Die Zeilen 35 bis 38 bilden einen Abfrageblock für die vier Funktionstasten <F1>, <F3>, <F5> und <F7>.

Wenn ihr ASCII-Code auftritt, dann drucken sie das Zeichen des hinter dem jeweiligen CHR$-Befehl stehenden Codewertes aus. Da diese Codewerte aber die Werte der Steuerzeichen – der Reihe nach – GELB, HELLBLAU, CURSOR HOME und BILDSCHIRM LÖSCHEN sind, werden diese Funktionen ausgeführt.

Das ist also das Kochrezept zur Nutzbarmachung der Funktionstasten. Aber auch alle anderen Tasten können so abgefragt und zur Steuerung irgendwelcher Programmschritte verwendet werden. Wir werden diese Methode auch noch benützen.

Übrigens, wenn Sie nur eine einzige Taste abfragen, deren Codewert aber nicht ausdrucken beziehungsweise nicht weiterverwenden, geht die Abfrage von oben dadurch schneller, daß Sie den String gar nicht erst lange umwandeln:

```
→ 10 GET A$: IF A$="" THEN 10
  35 IF A$=CHR$(133) THEN PRINT CHR$(150)
```

Die Prüfzeile 35 vergleicht einfach den eingegebenen String A$ mit dem String, der dem ASCII-Code 133 entspricht – der CHR$-Befehl macht das möglich. Nur bei vielen Abfrage-IFs ist das Tippen der vielen CHR$ mühsam und die ASC-Methode von vorher hat ihre Berechtigung.

```
10 PRINT"**** KODIEREN/DEKODIEREN ****    <206>
20 :                                      <252>
30 :                                      <006>
100 GET W$:IF W$="" THEN 100              <160>
110 IF W$=CHR$(133) THEN 270              <068>
120 PRINT W$;                             <065>
130 A$=A$+W$                              <248>
140 IF W$=CHR$(13) THEN 170               <142>
150 GOTO 100                              <078>
160 REM----------------------------       <253>
170 X=X+1                                 <198>
180 A=ASC(MID$(A$,X,1))                   <070>
190 B=A+X                                 <219>
200 B$=CHR$(B)                            <105>
210 C$=C$+B$                              <123>
220 IF X<>LEN(A$)-1 THEN 170              <233>
230 C$=C$+CHR$(32)                        <011>
240 PRINT C$                              <178>
250 A$=""                                 <023>
260 X=0:GOTO 100                          <208>
265 REM-----------------------------      <254>
270 PRINT"DEKODIEREN"                     <014>
280 INPUT E$                              <156>
285 IF E$="ENDE" THEN END                 <191>
290 X=X+1                                 <064>
300 E=ASC(MID$(E$,X,1))                   <212>
310 D=E-X                                 <189>
320 D$=D$+CHR$(D)                         <246>
330 IF X<>LEN(E$) THEN 290                <117>
340 D$=D$+CHR$(32)                        <159>
350 PRINT D$                              <042>
360 E$=""                                 <151>
370 X=0:GOTO 280                          <199>
© 64'er
```

Listing 2. Codieren und Decodieren mit den String-Befehlen



17 Daten aus dem Keller holen

Es gibt Situationen im Programmierleben, in denen es wünschenswert wäre, auf einen ganzen Stall voll Zahlen oder Wörter zurückgreifen zu können, ohne sie jedesmal per INPUT oder GET mühsam in den Computer eingeben zu müssen. Einmal eingegeben, sollten sie immer zur Verfügung stehen.

Diese Anforderung zeigt deutlich in Richtung »Speicherung von Daten« im Computer. Nun, wenn wir einer String-Variablen in einer Befehlszeile einen String zuordnen, wird dieser bekanntlich auch gespeichert. Aber das würde dazu führen, daß wir beispielsweise fünf Zuordnungen machen müßten, um fünf Strings zu speichern, etwa so:

```
→ 10 A$="BUCH"
  20 B$="SCHULTER"
  30 C$="SCHIFF"
  40 D$="TISCH"
  50 E$="MALEREI"
```

Bei fünf Strings geht es noch, aber bei 30 oder gar bei 200? Diesen Notstand stellt Basic ab mit dem Befehl

DATA

Hinter diesem Befehl können in einer Befehlszeile so viel Zahlenwerte oder Strings geschrieben und dadurch gespeichert werden, wie in einer Programmzeile Platz haben.

Alle Eintragungen in einer DATA-Zeile müssen durch Kommata getrennt sein.

Das Beispiel oben, das Sie bitte vor der Eingabe der nächsten Zeilen mit NEW löschen sollten, sieht jetzt so aus:

```
→ 10 DATA BUCH,SCHULTER,SCHIFF,TISCH,MALEREI
```

Sie sehen, die Strings brauchen nicht in Anführungszeichen stehen, mit einer Ausnahme. Diese ist weiter unten im Kasten beschrieben.



Eine DATA-Zeile mit Zahlen sieht so aus:

```
→ 20 DATA 25,123,225,16,24
```

Jetzt muß noch geklärt werden, wie man eigentlich die gespeicherten Daten herausholen kann.

Nun, das geht erstens immer der Reihe nach, und zweitens mit dem Basic-Befehl

READ

was leicht und treffend mit »LESEN« übersetzbar ist. Probieren wir es:

```
→ 40 READ A$
  50 PRINT A$
```

Nach RUN druckt das Programm das erste der Wörter in der DATA-Zeile, nämlich »BUCH« aus. An die anderen kommen wir nicht heran.

Wenn Sie in Zeile 40 und 50 die Variable in eine numerische Variable verwandeln, kommen wir trotzdem nicht an die Zahlen. Zusätzlich bestraft uns der Computer mit einer Fehlermeldung.

Bauen wir also eine Schleife ein, um den READ-Befehl fünfmal zu wiederholen.

```
→ 30 FOR X=1 TO 5
  60 NEXT
```

Jetzt tauchen alle fünf Wörter auf. Eine Erhöhung der Schleifenvariablen X auf 6 quittiert der Computer nach RUN wieder mit einer Fehlermeldung, da wir mit der Stringvariablen A$ in den Bereich der Zahlen eingedrungen sind, wodurch Variablentyp und Wert nicht zusammenpassen.

Eine zweite Schleife schafft Abhilfe:

```
→ 70 FOR X=1 TO 5
  80 READ A
  90 PRINT A
  100 NEXT
```

Wenigstens soweit haben wir es gebracht, daß wir alle gespeicherten Daten aus dem Speicher holen können.

Erproben Sie noch durch eine Erhöhung des Endwertes der Schleife in Zeile 70 auf 10, was passiert, wenn wir mehr Daten READen wollen, als vorhanden sind. Sie werden merken, daß das nicht geht, weniger dagegen schon.

Wir wollen uns jetzt auf nur einen Datentyp beschränken. Löschen Sie bitte die Zeilen 70 bis 100.

Ein weiteres Experiment besteht darin, den READ-Vorgang endlos zu wiederholen mit einer neuen Zeile 70.

```
→ 10 DATA BUCH,SCHULTER,SCHIFF,TISCH,MALEREI
  30 FOR X=1 TO 5
  40 READ A$
  50 PRINT A$
  60 NEXT
  70 GOTO 30
```

Auch dieser Versuch schlägt fehl.

Der Grund dafür liegt im Verfahren des READ-Befehls, nämlich einen internen Zähler mitlaufen zu lassen, der anzeigt, auf das wievielte Wort der READ-Befehl zugreifen muß. Und im obigen Fall ist dieser Zähler durch die Wiederholung mit GOTO 30 über die vorhandenen fünf Wörter hinausgelaufen.

Ein zu READ-DATA gehörender Basic-Befehl stellt diesen Zähler wieder an seinen Anfang zurück. Der Befehl lautet:

RESTORE

was soviel heißt wie »wiederherstellen«. Oben vor den GOTO-30-Befehl gestellt, setzt RESTORE in der Tat alles zurecht:

```
→ 65 RESTORE
```

Die jetzt erreichte Endlos-Schleife kann mit der STOP-Taste abgewürgt werden.

Ein lustiges Experiment will ich Ihnen noch zeigen, bevor wir ein weiteres Programm angehen.

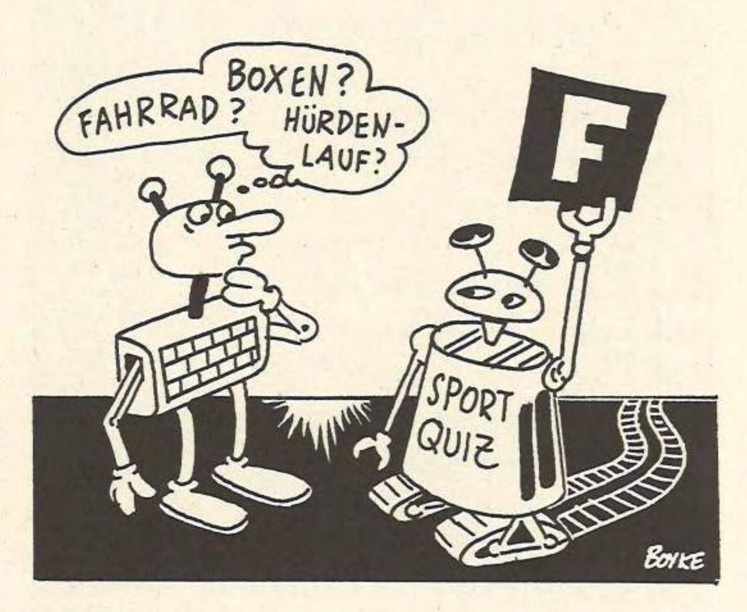
Basic-Befehle Nr. 23, 24 und 25 READ, DATA und RESTORE
- mit DATA können beliebig viele Zahlen und Strings in einem Programm gespeichert werden
- sie stehen hinter dem DATA-Befehl, durch Kommata voneinander getrennt
- Strings brauchen nicht zwischen Gänsefüßchen stehen, es sei denn, sie enthalten als Teil des Strings ein Komma oder ein Semikolon. Da diese besondere Steuerzwecke erfüllen, muß in diesem Fall der String zwischen Gänsefüßchen gesetzt werden
- mit dem Befehl READ werden die Daten der Reihe nach gelesen. Datentyp und Variablentyp müssen einander entsprechen
- sollen die Daten mehrfach ausgelesen werden, muß mit RESTORE der Anfangszustand des Auslesens hergestellt werden

Schreiben Sie die FOR-NEXT-Zeile 30 so wie angegeben und vertauschen Sie NEXT mit PRINT A$.

```
→ 30 FOR X=1 TO INT(RND(0)*5)+1
  40 READ A$
  50 NEXT
  60 PRINT A$
  70 RESTORE
  80 GOTO 30
```

Zeile 30 müßte Ihnen bekannt vorkommen. Die obere Grenze der Schleife wird jetzt per Zufall bestimmt. Sie kann nach unserem Kochrezept von früher die Werte 1 bis 5 annehmen, aber wie gesagt, vom Zufall bestimmt.

Da der PRINT-Befehl erst nach der Schleife kommt, druckt er nicht alle Wörter, sondern nur das zuletzt gelesene aus. Wir können somit eine zufällige Auswahl aus der Menge der gespeicherten Wörter treffen, was wir gleich beim nächsten Programm verwenden werden.



18 Ein Programm mit READ-DATA

Das Programm stammt ebenfalls aus dem schon mehrfach zitierten Buch »Computerspiele und Knobeleien« von Rüdiger Baumann.

Aufgabe:
- Der Computer hat viele Begriffe gespeichert. In unserem Fall sind es 14 Sportarten.
- Per Zufallsgenerator wird einer der Begriffe ausgewählt. Nur seine Länge wird dem Spieler durch Punkte bekanntgegeben.
- Der Spieler rät einen Buchstaben.
- Das Programm vergleicht diesen Buchstaben mit allen Buchstaben des Begriffes und schreibt den Buchstaben auf die richtigen Plätze, falls es eine Übereinstimmung feststellt.
- Der Vorgang wird so lange wiederholt, bis das Wort erraten ist.
- Zum Abschluß gibt der Spieler das komplette Wort ein.

Ich möchte das in Listing 3 abgedruckte Programm blockweise mit Ihnen durchgehen.

Zeile 10 löscht den Bildschirm - diesmal mit dem ASCII-Code!

Die Zeilen 80 bis 180 enthalten mit PRINT-Befehlen die Spielanleitung.

In den DATA-Zeilen 730 bis 760 sind die Sportarten gespeichert.

In den Zeilen 240 bis 260 wird in der vorher schon verwendeten Art und Weise per Zufall eine Sportart U$ ausgewählt.

Zeile 290 mißt die Länge des Wortes U$. Die Zeile 300 erfindet ein Hilfswort H$, welches am Anfang nur aus Punkten besteht, und zwar aus genauso vielen, wie der Sportbegriff lang ist.

Jetzt folgt der erste Versuch.

Zeile 370 druckt als Eingabebereitschaftszeichen (»Prompt« genannt) das punktierte Hilfswort H$ aus, um die Länge anzuzeigen.

In Zeile 390 wird per INPUT ein Buchstaben L$ angefordert und eingegeben. Wird gleich das ganze Wort eingegeben und ist es richtig, springt die Zeile 400 auf den Abschnitt der Entscheidung über Spielwiederholung ab Zeile 600.

Zeile 410 ist die Sicherung gegen unbeabsichtigtes Drücken der RETURN-Taste ohne Eingabe.

Der schwierigste Teil des Programms ist der Vergleich des eingegebenen Buchstabens mit dem vorgegebenen Sportbegriff. Er beginnt ab Zeile 610.

Zeile 480 und 530 bilden eine Zählschleife in der Länge des vorgegebenen Sportbegriffs U$. Innerhalb dieser Schleife holt Zeile 490 die einzelnen Buchstaben aus dem Wort U$ heraus - mit der Methode des MID$-Befehls. Zeile 500 macht dasselbe im Gleichtakt für das (gepunktete) Hilfswort H$.

Zeile 500 vergleicht jetzt zuerst den eingegebenen - geratenen - Buchstaben L$ mit jedem einzelnen Buchstaben B$ des Wortes U$. Ist er identisch, wird ein neues Hilfswort N$ aus dem richtig geratenen Buchstaben L$ gebildet. War der Buchstabe falsch, dann wird aus dem Hilfswort H$ ein Punkt in das neue Hilfswort N$ übernommen.

Das wird für alle Stellen der gleich langen Wörter U$ und H$ gemacht.

Danach wird in Zeile 540 das Hilfswort N$, das jetzt vielleicht schon eine Kombination aus Punkten und Buchstaben ist, dem alten Hilfswort H$ zugeordnet, welches ja dem Spieler beim nächsten Ratevorgang gezeigt wird (Zeile 370).

Danach wird der Ratevorgang in Zeile 390 und damit der ganze Ablauf wiederholt.

Jedesmal, wenn ein Buchstabe richtig geraten ist, kommt er an die entsprechende Stelle der Hilfswörter N$ und H$. Erst wenn der Spieler das ganze Wort kennt und es komplett in Zeile 390 eingibt, springt, wie erwähnt, Zeile 400 nach Zeile 600, wo von 630 bis 670 in üblicher Manier die Frage nach Wiederholung des Spiels oder Ende gestellt wird.

Alle anderen Zeilennummern dienen der Lesbarkeit und der Schönschrift des Programms.

```
10 PRINT CHR$(147)                                <039>
20 PRINT"***** BEGRIFFE RATEN ******              <211>
30 PRINT                                          <132>
40 :                                              <016>
50 REM-------- SPIELANLEITUNG----------           <206>
60 :                                              <036>
70 :                                              <046>
80 PRINT"DER COMPUTER DENKT SICH EINE             <155>
90 PRINT"SPORTART.                                <175>
100 PRINT"SIE SOLLEN SIE ERRATEN.                 <138>
110 PRINT                                         <212>
120 PRINT"SIE KOENNEN EINEN BUCHSTABEN            <097>
130 PRINT"EINGEBEN, DANN ERSCHEINEN               <187>
140 PRINT"DIE GERATENEN BUCHSTABEN AN             <076>
150 PRINT"DER RICHTIGEN STELLE.                   <165>
160 PRINT                                         <006>
170 PRINT"HABEN SIE DAS WORT ERRATEN,             <021>
180 PRINT"GEBEN SIE ES KOMPLETT EIN.              <169>
190 :                                             <166>
200 :                                             <176>
210 REM------- WORT AUSWAEHLEN -------            <243>
220 :                                             <196>
230 :                                             <206>
240 FOR X=1 TO INT(RND(0)*14)+1                   <155>
250 READ U$                                       <004>
260 NEXT                                          <016>
270 :                                             <248>
280 H$=""                                         <083>
290 FOR X=1 TO LEN(U$)                            <242>
300 H$=H$+" "                                     <008>
310 NEXT                                          <066>
320 :                                             <042>
330 :                                             <052>
340 REM-------- RATEVERSUCH ---------             <160>
350 :                                             <072>
360 PRINT:PRINT                                   <058>
370 PRINT"GESUCHT WIRD: ";H$                      <073>
380 PRINT                                         <228>
390 INPUT"WAS RATEN SIE";L$                       <220>
400 IF L$=U$ THEN 600                             <117>
410 IF LEN(L$)>1 THEN 360                         <067>
420 :                                             <142>
430 :                                             <152>
440 REM--------- HILFSWORT ----------             <065>
450 :                                             <172>
460 :                                             <182>
470 N$=""                                         <041>
480 FOR X=1 TO LEN(U$)                            <176>
490 B$=MID$(U$,X,1)                               <196>
500 C$=MID$(H$,X,1)                               <076>
510 IF L$=B$ THEN N$=N$+L$:GOTO 530               <010>
520 N$=N$+C$                                      <067>
530 NEXT                                          <032>
540 H$=N$                                         <095>
550 GOTO 360                                      <082>
560 :                                             <028>
570 :                                             <038>
580 REM---- NOCH EINMAL ? ----------              <043>
590 :                                             <058>
600 PRINT                                         <194>
610 PRINT"SIE HABEN RICHTIG GERATEN               <116>
620 PRINT                                         <214>
630 PRINT"NOCH EINMAL (J/N) ?                     <096>
640 GET A$: IF A$="" THEN 640                     <166>
650 IF A$="J" THEN RUN                            <166>
660 PRINT                                         <254>
670 PRINT"AUF WIEDERSEHEN                         <006>
680 :                                             <148>
690 :                                             <158>
700 REM------- WOERTERSPEICHER--------            <211>
710 :                                             <178>
720 :                                             <188>
730 DATA FUSSBALL,HOCKEY,GOLF,BOXEN               <115>
740 DATA TURNEN,SCHWIMMEN,HOCHSPRUNG              <099>
750 DATA SEGELN,FECHTEN,JUDO,BASKETBALL           <137>
760 DATA KEGELN,SCHIFAHREN,TENNIS                 <102>
© 64'er
```

Listing 3. Ein kleines Spiel

Weil anschließend ein völlig neues Thema angeschnitten wird, wäre es vielleicht wieder Zeit für ein Päuschen.

19 Variable in Feldern (Arrays)

Ich lade Sie zu einem Experiment ein. Wir stellen uns vor, wir benötigten in einem Programm eine Tabelle mit Zahlen, zum Beispiel die Anzahl der Bewohner, die in zehn einstöckigen Häusern in einer langen Straße wohnen. Die Adresse der Häuser unterscheidet sich nur durch die Hausnummer, denn der Straßenname bleibt ja gleich. Statt des langen Straßennamens nehmen wir nur einen Buchstaben, zum Beispiel T. Zur Kennzeichnung der zehn verschiedenen Adressen hängen wir die Ziffern 0 bis 9 hinten an. Die Zahl 10 können wir nicht nehmen, weil nach Punkt 3 des obigen Fazits die zweistelligen Variablennamen T1 und T10 identisch sind.

In unserem Beispiel wollen wir den einzelnen Häusern Bewohnerzahlen zuweisen, die jeweils um 2 größer sind als die Hausnummer.

```
10 T0=2
20 T1=3
30 T2=4
40 T3=5
etc.
100 T9=11
110 PRINT T0;T1;
etc. bis T9
```

Das ist natürlich eine sehr umständliche Arbeit, und langweilig ist diese dauernde Wiederholung obendrein.

Also, wie geht das einfacher und eleganter? Natürlich mit einer FOR-TO-NEXT-Schleife, aber aufgepaßt: bei ihrer Verwendung müssen wir die Hausnummer N in Klammern dem Straßennamen T anhängen. Anders akzeptiert das der C64 in Basic nicht.

Als Programm sieht das so aus:

```
10 FOR N=0 TO 9
20 T(N)=N+2
30 PRINT T(N)
40 NEXT N
```

Ihnen ist natürlich klar, daß Straßenname T plus Hausnummer (N) eine Variable ist, der wir Werte (Bewohner) zuweisen. Aber eins sollte Sie stutzig machen. Wie unterscheiden sich diese speziellen Variablen voneinander? Sind T(3) und T(8) nicht identisch, wenn nach der oben schon zitierten Regel nur die ersten beiden Zeichen des Namens - hier T(- hergenommen werden ?

Nun, die Klammer macht den Unterschied. Durch sie wird ein neuer Typ einer Variablen, eine sogenannte Feld-Variable definiert, und Feld-Variablen unterscheiden sich durch die Zahl innerhalb der Klammer. Diese Zahl heißt Index (Mehrzahl: Indizes). Warum sie ausgerechnet Feld-Variable heißt, erkläre ich gleich.

Vorher, bei der Schreibweise T0, T1 etc. waren wir auf 10 Variable beschränkt, weil T1 und T10 dieselbe Variable war. Jetzt - mit Indizes als Unterscheidungsmerkmal - gilt diese Beschränkung nicht. Deshalb wollen wir die Anzahl der Indizes im Programm noch erhöhen. Bei einem N von 0 bis 10 geht es ohne Probleme. Aber ab 11 ist schon wieder Feierabend. Wir erhalten die Fehlermeldung »BAD SUBSCRIPT ERROR IN 20«. Um Ihnen das zu erklären, muß ich genauer beschreiben, was passiert, wenn wir eine Variable mit einem Index in Klammern verwenden.

Wenn wir eine Variable so schreiben:

T(1)=25

dann nimmt der Computer an, daß wir in einer Tabelle noch mehrere derartige Variable T( ) verwenden wollen - klar, sonst würden wir uns ja die Mühe mit der Klammer nicht machen. Um uns die Sache zu erleichtern, reserviert der Computer unter dem Variablennamen T in seinem Speicher von vornherein elf Plätze, für T(0) bis T(10). Diesen reservierten Bereich können Sie salopp Tabelle nennen; offiziell heißt er *Feld* oder auf englisch *Array*. Die alte Regel für den Namen der Variablen gilt jetzt auch nicht mehr, denn zum Unterscheiden der einzelnen Feld-Variablen desselben Anfangsbuchstabens bedient der Computer sich der Index-Zahl in der Klammer.

Die Beschränkung auf eine Feld-(Array-)Größe von elf Plätzen wäre natürlich sehr lästig, wenn sie nicht umgangen werden könnte. Wenn wir nämlich mehr Platz brauchen, können wir dem Computer mit dem Basic-Befehl

DIM

unsere Reservierungswünsche mitteilen. DIM ist eine Abkürzung, die aus dem Wort »Dimension« abgeleitet ist. Als Beispiel möge die folgende Programmzeile dienen:

```
→ 5 DIM T(25)
```

Sie reserviert im obigen Programm für die Variable T( ) im Speicher ein Feld von 26 Plätzen, von T(0) bis T(25).

Die Größe eines Feldes ist nur durch den vorhandenen Speicherplatz begrenzt. Wenn Sie die Zahl zu groß wählen, verweigert der Computer die Reservierung - natürlich erst nach dem Befehl RUN - mit OUT OF MEMORY, was soviel heißt wie »keinen Platz mehr im Speicher«.

Basic-Befehl Nr. 26 DIM

- dieser Befehl wird in der folgenden Schreibweise verwendet: DIM Variablenname (Index)
- er reserviert einen Speicherbereich für eine durch den Index festgelegte Anzahl von Variablen desselben Namens, die sich nur durch ihren jeweiligen Index unterscheiden. Dieser Speicherbereich wird Feld oder Array genannt
- mit DIM können sowohl Felder für numerische als auch für String-Variable reserviert werden. Ein Feld kann immer nur einen einzigen Variablentyp enthalten
- für den Namen und für die Kennzeichnung des Typs der Feld-Variablen (vor der Klammer) gelten dieselben Regeln wie für alle anderen Variablen

Ein Feld kann also auch aus String-Variablen bestehen:

```
→ 200 DIM A$(25)
  210 A$(0)="FEUER"
  220 A$(1)="ZANGEN"
  230 A$(2)="BOWLE"
      .        .
      .        .
      .        .
  300 FOR N=0 TO 2
  310 PRINT A$(N);
  320 NEXT N
```

Dieses kleine Programm reserviert ein Feld von 25 Strings und weist den ersten dreien davon je ein Wort zu. Mit RUN 200 gestartet, druckt es das Wort Feuerzangenbowle aus, indem in der Zeile 310 nacheinander für die Werte von N=0 bis N=2 die gespeicherten Strings A$(0) bis A$(2) durch das Semikolon nach dem PRINT-Befehl aneinandergeklebt werden. Das Eintragen der einzelnen Werte in die Tabelle, das ich in den Zeilen 210 bis 230 vorgenommen und danach nur noch angedeutet habe, geht viel eleganter mit den Befehlen READ-DATA, die in der Lektion 17 bereits ausführlich beschrieben sind.

Um das zu zeigen, schlage ich Ihnen eine kleine Anwendung - im Computerdeutsch heißt so ein Programm

»Utility« – vor, und zwar eine Nachschlagliste von Geburtstagen Ihrer Freunde und Verwandten. Natürlich ist das keine Utility, deretwegen ich mir einen Computer kaufen würde, aber für unsere Zwecke hier ist sie ganz passend.

Ich lege das Beispiel vorerst einmal auf fünf Namen (N$) und dazugehörige Geburtstagsdaten (D$) aus. Wir brauchen also zwei Felder mit je fünf Plätzen, dazu zwei DATA-Zeilen mit den Eintragungen. Verwenden Sie bitte meine wirr erscheinenden Zeilennummern; am Ende passen sie schon zusammen.

Speicher löschen mit NEW

```
→ 20 DIM N$(4)
  500 DATA MAX,MORITZ,MARIA,HANS,LUISE
  120 DIM D$(4)
  600 DATA 12.6.52,3.4.60,21.1.40,19.9.56,11.11.
  70
```

Denken Sie bitte daran: DIM N$(4) legt fünf Plätze fest, nämlich 0 bis 4.

Denken Sie ebenfalls daran, daß Eintragungen in DATA-Zeilen durch ein Komma voneinander getrennt sein müssen.

So, jetzt brauchen wir für beide Felder eine Schleife, mit der die Eintragungen der DATA-Zeilen mit READ in das jeweilige Feld gelesen werden. Für das Namensfeld gilt:

```
→ 30 FOR I=0 TO 4
  40 READ N$(I)
  50 NEXT I
```

Dasselbe machen wir für das zweite Feld der Geburtstage:

```
→ 130 FOR I=0 TO 4
  140 READ D$(I)
  150 NEXT I
```

Jetzt brauchen wir noch einen Programmteil, der nach Eingabe eines Namens das dazugehörige Geburtsdatum heraussucht und ausdruckt. Für die »Dazugehörigkeit« benützen wir die Indizes der Feldvariablen. Das bedeutet nichts anderes, als daß zum zweiten Namen N$(2) das zweite Datum D$(2) gehört.

Doch zuerst stellen wir per INPUT die Frage F$ nach dem Namen, dessen Geburtstag wir wissen wollen:

```
→ 200 INPUT "NAME";F$
```

Dann vergleichen wir in einer weiteren Schleife die gespeicherten Namen N$(I) mit dem gefragten Namen F$. Wenn bei einem bestimmten Wert I der Vergleichsschleife die beiden Namen gleich sind, wird der Geburtstag mit demselben Index I ausgedruckt. Das Programm kann dann eine weitere Eingabe eines Namens verlangen (GOTO 200).

```
→ 210 FOR I=0 TO 4
  220 IF N$(I)=F$ THEN PRINT D$(I):GOTO 200
  230 NEXT I
```

Um den Vergleichs- und Entscheidungsvorgang genau zu verstehen, brauchen Sie lediglich die Schleifen durchzuspielen, indem Sie die Werte von I gedanklich hochzählen und im Programm nachschauen, was passiert. Wir können aber zum Verständnis auch eine »Lehrzeile« (mit h !) einfügen, die uns den jeweiligen Stand des Vergleiches anzeigt:

```
→ 215 PRINT I;F$,N$(I);D$(I)
```

Nach RUN und nach Eingabe eines der fünf Namen sehen wir am Bildschirm eine Reihe mit steigenden Werten von I, daneben (Semikolon !) den eingegebenen Namen, dann in einem größeren Abstand (Komma !) die Namen und Daten, und zwar so lange, bis F$ und N$ gleich sind.

Nehmen Sie jetzt Zeile 215 wieder heraus, aber fügen Sie bitte eine Zeile 240 dazu, die wir für ein bedienungsfreundliches Programm brauchen. Wir müssen nämlich Vorkehrung treffen für den Fall, daß ein Name eingegeben wird, der nicht in der Liste steht. Auch Tippfehler fallen in diese Kategorie. Wenn kein positiver Vergleich zwischen F$ und N$ innerhalb der Schleife auftritt, macht das Programm nach der Schleife weiter, mit der neuen Zeile 240, die für sich selbst spricht:

```
→ 215
  240 PRINT"NAME IST NICHT IN DER LISTE"
```

Das ganze Programm sieht so aus:

```
→ 20 DIM N$(4)
  30 FOR I=0 TO 4
  40 READ N$(I)
  50 NEXT I
  120 DIM D$(4)
  130 FOR I=0 TO 4
  140 READ D$(I)
  150 NEXT I
  200 INPUT "NAME";F$
  210 FOR I=0 TO 4
  220 IF N$(I)=F$ THEN PRINT D$(I):GOTO 200
  230 NEXT I
  240 PRINT"NAME IST NICHT IN DER LISTE"
  500 DATA MAX,MORITZ,MARIA,HANS,LUISE
  600 DATA 12.6.52,3.4.60,21.1.40,19.9.56,11.11
  .70
```

Der DIM-Befehl und der READ-Befehl haben beide eine zusätzliche Eigenschaft, die uns eine wesentliche Verbesserung dieses Programms erlaubt:

Man darf hinter einem einzigen DIM-Befehl mehrere Felder dimensionieren. Sie müssen lediglich durch ein Komma getrennt sein. Genauso darf man mit einem einzigen READ-Befehl mehrere Werte-Gruppen auslesen. Hier gilt dieselbe Kommaregel wie bei DIM.

Damit können wir die Zeilen 20 bis 150 stark verkürzen:

```
→ 20 DIM N$(4),D$(4)
  30 FOR I=0 TO 4
  40 READ N$(I),D$(I)
  50 NEXT I
```

Die Zeile 20 ist einfach nur eine Zusammenziehung der beiden alten Zeilen 20 und 120. Die Funktion des DIM-Befehls bleibt dabei erhalten – er dimensioniert halt gleich beide Felder.

Beim »doppelten« READ-Befehl in Zeile 40 hat sich in der Arbeitsweise, verglichen mit den alten Zeilen 40 und 140, etwas geändert. Schuld daran ist aber die FOR-NEXT-Schleife der Zeile 30.

Bei jedem Wert von I holt nämlich der READ-Befehl zwei hintereinanderliegende Werte aus den DATA-Zeilen und weist sie den beiden Variablen N$ und D$ zu. Wir müssen diesem Vorgang dadurch Rechnung tragen, daß wir den Inhalt der DATA-Zeilen umorganisieren. Es müssen jetzt immer je ein Name und das dazugehörige Geburtsdatum hintereinander kommen, was eigentlich viel leichter einzutippen ist. Sie sehen, gute Programmierung ist fast immer auch klarer und logischer.

```
→ 500 DATA MAX,31.3.55,MORITZ,12.4.45,
  HANS,6.2.60
  600 DATA MARIA,14.7.63,LUISE,8.9.60
```

Mit dieser Anordnung haben wir noch einen weiteren Vorteil erhalten. Das Programm, oder besser gesagt die Geburtstagskartei, kann ganz leicht erweitert werden. Einen neuen Namen samt Datum brauchen Sie nur durch Komma getrennt hinter die letzte Eintragung der DATA-Zeilen zu schreiben.

Doch halt! Noch etwas fehlt: Die Zahl der Namen, die sich durch eine neue Eintragung natürlich erhöht, kommt ja im Programm auch vor, und zwar in unserem Fall als Ziffer 4 in den Zeilen 20, 30 und 210. Um uns diese Zusatzarbeit ebenfalls zu erleichtern, geben wir dieser Zahl einen Variablen-Namen Z, definieren dieses Z ganz am Anfang

Merken wir uns:

1. Hinter einem DIM-Befehl können mehrere Felder dimensioniert werden. Sie müssen lediglich durch ein Komma getrennt sein.
2. Mit einem READ-Befehl können mehrere DATA-Werte gelesen werden. Auch diese müssen durch ein Komma voneinander getrennt werden.
3. Es empfiehlt sich, einen Zahlenwert oder einen String, der mehrmals in einem Programm vorkommt, ganz am Anfang des Programms als numerische beziehungsweise als Stringvariable zu definieren und im Programm dann nur einmal dieser Variablen den Wert zuzuweisen.

und brauchen so bei jeder neuen Namenseintragung nur dieses Z um 1 erhöhen.

Diese Zeilen sehen jetzt so aus:

```
→ 10 Z=4
  20 DIM N$(Z),D$(Z)
  30 FOR I=0 TO Z
  210 FOR I=0 TO Z
```

Wir haben jetzt ein komplettes kleines Programm, dessen zusammengewürfelte Zeilennummern ihm allerdings ein unfertiges Aussehen verleihen.

Deswegen und wegen der vielen Änderungen während seiner Entstehung ist es im folgenden Listing 4 noch einmal komplett dargestellt.

```
10 Z=4                                          <075>
20 DIM N$(Z),D$(Z)                              <096>
30 FOR I=0 TO Z                                 <245>
40 READ N$(I),D$(I)                             <035>
50 NEXT I                                       <134>
200 INPUT "NAME";F$                             <194>
210 FOR I=0 TO Z                                <169>
220 IF N$(I)=F$ THEN PRINT D$(I):GOTO 200       <246>
230 NEXT I                                      <058>
240 PRINT "NAME IST NICHT IN DER LISTE          <249>
500 DATA MAX,31.3.55,MORITZ,12.4.45,HANS,6
 .2.60                                          <186>
600 DATA MARIA,14.7.63,LUISE,8.9.60             <196>
```

© 64'er

Listing 4. Eine einfache Geburtstagsverwaltung

20 Zweidimensionale Felder (Arrays)

Hinter dieser Überschrift verbirgt sich eine Erweiterung des DIM-Befehls.

- eindimensional ist eine Linie; sie hat nur in einer Richtung eine Ausdehnung
- zweidimensional ist eine Fläche; sie hat eine Länge und eine Breite.

Diese Betrachtungsweise kann auch auf Variablenfelder angewendet werden.

Ich habe bei der ersten Erklärung des Begriffs »Feld« und »Feldvariable« den Vergleich mit einstöckigen Häusern in einer Straße verwendet. Unser damaliges Beispiel könnte man in dieser Form aufmalen:

Bewohner	2	3	4	5	6
Adresse	T(0)	T(1)	T(2)	T(3)	T(4)

Ein zweidimensionales Feld ist eine Tabelle, in der sowohl waagerecht wie senkrecht Eintragungen möglich sind:

Stockwerk 3	T(0,3)	T(1,3)	T(2,3)	T(3,3)	T(4,3)
2	T(0,2)	T(1,2)	T(2,2)	T(3,2)	T(4,2)
1	T(0,1)	T(1,1)	T(2,1)	T(3,1)	T(4,1)
0	T(0,0)	T(1,0)	T(2,0)	T(3,0)	T(4,0)
Hausnummer	0	1	2	3	4

In unserem Beispiel sind das Häuser, die mehrere Stockwerke und pro Stockwerk verschiedene Bewohnerzahlen haben. Der Einfachheit halber gebe ich allen Häusern dieselbe Zahl von Stockwerken.

Bei diesen Reihenhäusern habe ich nicht mehr die Zahl der Bewohner in die Häuser beziehungsweise Stockwerke geschrieben, sondern die »Adressen«, die wieder unsere Feldvariablen sind. Nur haben sie diesmal zwei Indizes, einen für das Stockwerk (Zeile des Feldes) und einen für die Hausnummer (Spalte des Feldes).

Wenn wir jetzt sagen wollen, daß im 3. Stock des 1. Hauses sieben Leute wohnen, legen wir das fest mit:

```
→ T(1,3)=7
```

Bei der Definition einer zweidimensionalen Variablen T(A,B) reserviert auch diesmal wieder der Computer ein Feld von 11 Variablen, allerdings pro Index, das heißt insgesamt 11 * 11=121 Plätze.

Ein kleines Programm beweist diese Behauptung:

```
→ 10 FOR I=0 TO 10
  20   FOR K=0 TO 10
  30     T(I,K)=I+K
  40     PRINT T(I,K);
  50   NEXT K
  60 NEXT I
```

Ich habe es extra so geschrieben, daß Sie die beiden Schleifen besser sehen können. Pro Durchlauf der äußeren I-Schleife läuft die innere K-Schleife 11mal durch. Dementsprechend sieht das Resultat der Zeile 40 aus.

Wenn Sie jetzt die obere Grenze von I auf 11 erhöhen, merkt das Programm erst nach dem 121.Durchlauf, daß zuwenig Platz reserviert worden ist. Lassen Sie dagegen das I auf maximal 10, erhöhen aber das K auf 11, bleibt das Programm schon nach dem ersten Durchlauf der inneren Schleife stehen.

Das war eine kleine Erinnerung an die Wirkungsweise geschachtelter Schleifen.

Wenn wir ein größeres Feld benötigen, müssen wir diesen Bedarf wieder mit dem DIM-Befehl eingeben:

DIM T(25,34)

Um die Wirkungsweise eines zweidimensionalen Feldes vorzuführen, habe ich vor, mit Ihnen einen »Karteikasten« zu entwickeln, in dem Sie eine Literatursammlung, Geschichtsdaten, oder alle Titel Ihrer Plattensammlung kata-

Merken wir uns:

1. Felder können mehrere Dimensionen haben. Eine zweidimensionale Feld-Variable hat zwei Indizes in der Klammer, die durch ein Komma getrennt sein müssen.
2. Für eine mehrdimensionale Feld-Variable werden pro Index elf Plätze im Speicher reserviert.
3. Zur Dimensionierung größerer Felder steht der DIM-Befehl, ebenfalls mit zwei Indizes, zur Verfügung.
4. Für mehrdimensionale DIM-Befehle und Feld-Variable gelten dieselben Regeln wie für eindimensionale Felder.

logisieren können. Eine Kartei ist natürlich nur dann sinnvoll, wenn bestimmte Eintragungen schnell durch Angabe eines Stichwortes gefunden werden können.

Ich selbst habe mir für meine Arbeiten eine Computer-Literatursammlung zugelegt, deren Struktur ich hier verwenden will. Von jedem interessanten Artikel oder Buchkapitel mache ich folgende Eintragung:
- Zeitschrift (oder Buch)
- Titel des Aufsatzes
- Sachgebiet
- Datum der Veröffentlichung
- Seitennummer
- Computer-Typ

Diese Eintragungen nenne ich DATENSATZ, jeder Datensatz enthält sechs KATEGORIEN.

Diese sollen nun in einem zweidimensionalen Feld untergebracht werden.

KATEGORIE	K=1	K=2	K=3	K=4	K=5	K=6
DATENSATZ	ZEITSCHRIFT	TITEL	SACHGEBIET	DATUM	SEITE	COMPUTERTYP
D=1	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
D=2	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
D=3	(3,1)	(3,3)	usw.			

Der erste Index »D« gibt also die Nummer des Datensatzes an, der zweite Index »K« die jeweilige Kategorie.

Um fürs erste einmal 100 Datensätze zu je sechs Kategorien eintragen zu können, dimensionieren wir ein Feld A$:

```
110 DIM A$(100,6)
```

Die ersten fünf Datensätze könnten zum Beispiel so aussehen:

```
1002 DATA GAZETTE,SPEEDSCRIPT,UTILITY,
1/84,12,64
1003 DATA RUN,DER DIM-BEFEHL,BASIC,3/85,125,20
1004 DATA 64er,FILES,KURS,4/87,118,64
1005 DATA CHIP,SCHLEIFEN,BASIC,11/83,134,20/64
1006 DATA HAPPY,GARBAGE COLLECTION,KURS,
4/87,20/64
```

Diese Datensätze in DATA-Zeilen lesen wir mit den folgenden Zeilen in das Feld A$:

```
120 D=D+1
130 FOR K=1 TO 6
140 READ A$(D,K)
150
160 NEXT K
170 GOTO 120
```

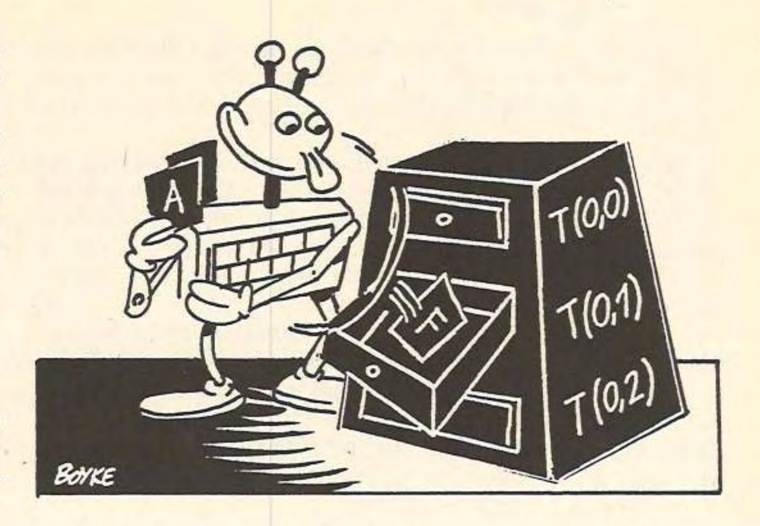
Da wir nicht wissen können, wie viele Datensätze wirklich im Feld enthalten sind, muß die D-Schleife hochgezählt werden, während die K-Schleife fest von 1 bis 6 läuft, da ja die Anzahl der Kategorien mit 6 festliegt.

Nun brauchen wir am Ende der Datensätze - ich lege es auf Zeile 2000 - eine Endmarkierung; sonst läuft der READ-Vorgang über. Ich wähle dafür den Klammeraffen »@«, dessen Auftreten wir in Zeile 150 abfragen:

```
150 IF A$(D,K)="@" GOTO 210
2000 DATA @
```

Ab Zeile 210 soll der Suchvorgang nach einem bestimmten Datensatz beginnen. Der Suchvorgang beginnt mit der Eingabe des Stichwortes S$. Es kann ein Sachgebiet sein, eine Zeitschrift oder ein bestimmter Titel, den Sie suchen. Auf alle Fälle soll das Programm alle Datensätze, die das Stichwort enthalten, ausdrucken.

```
210 INPUT"STICHWORT";S$
220 FOR S=1 TO D
230 FOR K=1 TO 6
```




```
240 IF A$(S,K)=S$ THEN FOR Z=1 TO 6:PRINT A$
(S,Z):NEXT Z
250 NEXT K
260
270 NEXT S
280 PRINT "ENDE DER SUCHE"
```

Um alle Datensätze, die im Feld A$ stehen, nach dem Stichwort S$ abzusuchen, bilden wir zwei verschachtelte Schleifen. Die eine zählt die Datensätze von 1 bis zum letzten Datensatz, dessen Nummer vom Einlesevorgang her noch auf dem Wert D steht. Für diese Schleife wähle ich die Variable S. Die andere Schleife zählt wieder die Kategorien K von 1 bis 6 (Zeilen 220 und 230).

In Zeile 240 wird pro Schritt geprüft, ob die Feldeintragung mit dem Stichwort S$ übereinstimmt.

Stimmt sie überein, dann soll der ganze Datensatz, in dem das Stichwort gefunden worden ist, ausgedruckt werden. Deswegen legen wir hinter das THEN sofort eine Sechserschleife zum Ausdrucken aller sechs Kategorien.

Danach wird die Suche mit den K- und S-Schleifen fortgesetzt, denn das Stichwort kann ja auch in nachfolgenden Datensätzen vorkommen.

Wenn Sie dieses Programmfragment laufenlassen, wird es aussteigen mit der Fehlermeldung »NEXT WITHOUT FOR«.

Der Fehler, der hier auftritt, ist so häufig und doch so schwer zu sehen, daß ich ihn extra hier eingebaut habe, um Sie daraufzustoßen.

Wir haben in den Zeilen 120 bis 150 eine Schleife mit der Variablen K für die sechs Kategorien. Dasselbe wiederholen wir in den Zeilen 230 bis 250. Und genau das geht schief, weil wir nämlich aus der ersten K-Schleife in Zeile 150 herausspringen, ohne daß die Schleife zu Ende gelaufen ist. Im Speicher des Computers steht also noch eine Schleifenvariable K auf einem bestimmten Wert, wenn wir in Zeile 230 eine neue K-Schleife anfangen. Das kann der Computer nicht verarbeiten.

Wir zählen übrigens die sechs Kategorien noch einmal in der Zeile 240, nur da macht es nichts, denn wir haben eine andere Schleifenvariable, nämlich Z verwendet.

Das ist also schon eine der möglichen Lösungen, nämlich eine Schleifenvariable immer nur einmal zu verwenden. Die sauberste Lösung ist aber, nach dem Aussprung die Schleife »künstlich« zum Ende zu bringen und das wollen wir in Zeile 150 auch machen. Verbessern Sie bitte:

```
150 IF A$(D,K)="@"THEN K=6:NEXT K:GOTO 210
```

Wir setzten also die Schleifenvariable K auf ihren Endwert 6 und geben ein letztes NEXT K. Damit wird die Schleife »geschlossen«, und wir dürfen ungestraft K als Schleifenvariable wiederverwenden.



Jetzt bleibt nur noch die Frage an den Benutzer, ob der Programmlauf wiederholt oder abgebrochen werden soll (Zeilen 310 bis 340). Das komplette Programm steht in Listing 5.

Diese »Kartei« kann durch Eintippen von zusätzlichen DATA-Zeilen beliebig vergrößert werden. Soll sie mehr als 100 Datensätze enthalten, dann muß in Zeile 110 das Feld entsprechend vergrößert dimensioniert werden. Letztlich ist das Programm nur begrenzt durch den verfügbaren Speicher des C64.

Das zweidimensionale Feld mit Variablen steht also im Speicher des C64 gespeichert. Wenn wir den Computer ausschalten, wird der Speicher gelöscht und alles ist weg. Es wäre deshalb schön, wenn wir das Feld mit den Daten auch auf ein Band oder auf eine Diskette abspeichern könnten, um es bei einer späteren Gelegenheit wieder in den Computer holen zu können.

```
100 REM**** DATEN INS FELD LESEN ****          <071>
105 :                                          <081>
110 DIM A$(100,6)                              <230>
120 D=D+1                                      <003>
130 FOR K=1 TO 6                               <119>
140 READ A$(D,K)                               <191>
150 IF A$(D,K)="@"THEN K=6:NEXT K:GOTO 210     <171>
160 NEXT K                                     <004>
170 GOTO 120                                   <130>
195 :                                          <171>
200 REM**** SUCHVORGANG *****                  <020>
205 :                                          <181>
210 INPUT"STICHWORT";S$                        <148>
220 FOR S=1 TO D                               <024>
230 FOR K=1 TO 6                               <219>
240 IF A$(S,K)=S$ THEN FOR Z=1 TO 6:PRINT
    A$(S,Z):NEXT Z                             <162>
250 NEXT K                                     <094>
260 PRINT                                      <108>
270 NEXT S                                     <180>
280 PRINT"ENDE DER SUCHE"                      <165>
295 :                                          <017>
300 REM**** WIEDERHOLUNG ODER ENDE ****        <101>
305 :                                          <027>
310 PRINT"NOCH EINMAL (J/N)?"                  <064>
320 GET V$:IF V$="" THEN 320                   <182>
330 IF V$="J" THEN 210                         <054>
340 END                                        <088>
999 :                                          <213>
1000 REM**** DATENSAETZE *****                 <166>
1001 :                                         <215>
1002 DATA GAZETTE,SPEEDSCRIPT,UTILITY,1/84
     ,12,64                                    <001>
1003 DATA RUN,DIM,BASIC,3/58,125,20            <044>
1004 DATA 64ER,FILES,KURS,4/87,118,64          <145>
1005 DATA CHIP,SCHLEIFEN,BASIC,11/83,134,2
     0/64                                      <066>
1006 DATA HAPPY,GARBAGE COLLECTION,KURS,4/
     87,38,20/64                               <082>
10000 DATA @                                   <110>
```


Listing 5. Zweidimensionales Feld

Aus dem Betriebshandbuch von Commodore kennen Sie sicher die Speicher- und Ladebefehle SAVE, LOAD für Kassetten und Disketten. Ich sage es aber besser gleich, bevor Sie unnütz experimentieren: Diese Befehle speichern und laden leider nur Programme, nicht aber Variablenwerte, die im Lauf eines Programms erst erzeugt werden.

Es gibt aber noch eine andere Methode, wie man Variablenwerte, sei es für numerische, sei es für Feldvariable, speichern kann. Dazu aber muß ich Ihnen erst in Lektion 21 ein bißchen mehr über den Speicher des C64 erklären. Speichern Sie bitte das Fragment Listing 5 auf Band oder Diskette ab, wir werden es später vervollständigen.

Der nun folgende Abschnitt hat nur indirekt etwas mit Basic zu tun. Aber wie ganz am Anfang betont, komme ich ohne einige Beschreibungen von Eigenschaften oder besser gesagt von Eigenheiten des C64 nicht aus. Einige Kenntnisse des Innenlebens des Computers gehören halt doch auch zum Programmieren.

Merken wir uns:
1. Schleifen, die durch Aussprung verlassen werden, bevor sie zu Ende gelaufen sind, werden im Speicher des Computers als »noch offen« registriert. Ihre Schleifenvariable kann daher nicht wiederverwendet werden.
2. Entweder muß man die Wiederverwendung von Schleifenvariablen vermeiden oder muß verlassene Schleifen schließen durch Zuweisung der Variablen auf den vorgesehenen Endwert plus nachfolgendem NEXT-Befehl.
3. Zweidimensionale Felder werden durch zwei verschachtelte Schleifen abgearbeitet.
4. Felder beziehungsweise die Werte der Feldvariablen können nicht mit SAVE auf Kassette oder Diskette gespeichert werden.



21 Der Speicher – das Gedächtnis des Computers

Alle Computer – Großrechenanlagen genauso wie kleine Heimcomputer – sind aus den folgenden Grundbausteinen aufgebaut:
- zentrale Recheneinheit, auch Mikroprozessor oder CPU (Central Processing Unit) genannt
- Speichereinheit
- Ein- und Ausgabe-Bausteine

Während die CPU rechnet, alle Vorgänge im Computer steuert, Befehle ausführt und somit das eigentliche Herz des Computers darstellt, braucht man die Ein- und Ausgabe-Bausteine, um Daten in den Computer hinein- beziehungsweise herauszuleiten. Alle Anschlüsse von Datasette, Diskettenlaufwerk, Drucker und Bildschirm werden von ihnen gesteuert.

Der Speicher ist der Notizblock des Computers. In ihm steht alles, was er sich merken soll: Programmzeilen, Variablenwerte, Zeichenketten (Strings) und so weiter. Auf englisch heißt Speicher passenderweise Memory, was wir wiederum auf deutsch mit Gedächtnis übersetzen könnten.

Jeder Computer hat zwei Arten von Speicher:

Das RAM (Random Access Memory) ist ein aus elektronischen Bauteilen aufgebauter Speicher, der für Programme frei verfügbar ist. Wir können in diesen Speichertyp Daten

hineinschreiben und sie wieder herauslesen, ohne sie zu zerstören. Nur nach dem Ausschalten des Computers - da sind sie weg!

Das passiert auch dann, wenn nur kurzzeitig der Strom ausfällt, etwa bei einem Gewitter oder wenn der Netzstecker einen Wackelkontakt hat.

Es ist deshalb ratsam, bei längeren Programmierarbeiten Zwischenergebnisse, das heißt den jeweiligen Inhalt des RAM auf Kassette oder Diskette abzuspeichern, denn nur dort sind sie dauerhaft sicher.

Das ROM (Read Only Memory) ist nicht frei verfügbar. Es besteht zwar auch aus elektronischen Bauteilen, aber sein Inhalt ist fest »eingebrannt« - man kann ihn nicht ändern. Er wird vom Computer selbst verwendet. Wenn Sie zum Beispiel Ihren C64 einschalten, dann läuft eine im ROM eingespeicherte Folge von Programmschritten ab, die schließlich mit der Meldung des Computers auf dem Bildschirm endet, mit der er sich bereit (READY) meldet. Im ROM stehen nicht nur die Programmschritte, die den Betrieb des Computers steuern, sondern auch das Programm, welches alle Basic-Befehle in einen Code »übersetzt«, den der zentrale Mikroprozessor (CPU) versteht. Wenn Sie mehr über die allgemeine Arbeitsweise des Computers erfahren wollen, dann lesen Sie bitte im 64'er-Sonderheft 38 nach.

21.1. Die Adressen der Speicherzellen

Der gesamte Speicher des C64 ist aus einzelnen Speicherzellen aufgebaut. Der C64 bietet Platz für 65536 Speicherzellen. Jede dieser Speicherzellen hat eine Nummer, von 0 bis 65535 (Bild 1).

Diese Nummern nennen wir Adressen. Um eine Zahl oder ein Zeichen in eine bestimmte Speicherzelle hineinschreiben zu können, müssen wir ihre Adresse kennen. Natürlich gilt dasselbe für das Auslesen. Es liegt deshalb sehr nahe, uns ein »Adreßbuch« des Speichers zu beschaffen.

Im Bedienungshandbuch des C 64 ist eine derartige Liste im Anhang auf den Seiten 160 bis 165 unter dem Titel »Speicherbelegung« angegeben (Beim C 128-Handbuch finden Sie diese Liste für den C 64-Modus im Anhang H). Diese Liste ist aber alles andere als klar und verständlich. In dem folgenden Absatz habe ich daher eine andere Darstellung gewählt, die Ihnen eine Übersicht über die verschiedenen Speicherbereiche und ihre Bedeutung geben soll. Aber ein Adreßbuch, oder wie es auf englisch heißt, eine »Memory Map« ist das eigentlich auch nicht. Dazu müßte ich ja jede einzelne Adresse beschreiben.

Hier ist also eine Kurzfassung eines Adreßbuches, in der wir aber bereits viel sehen können:

- Der Speicher beginnt ganz unten bei Adresse 0.
- Die ersten 2047 RAM-Speicherzellen sind vom Computer selbst belegt.
- Ab Adresse 2048 beginnt der RAM-Speicher für Basic-Programme. Ab hier werden alle eingegebenen oder von Band und Diskette geladenen Programme gespeichert. Auch alle Variablen, Felder und Zeichenketten, die im Lauf eines Programmes auftauchen, werden dort gespeichert.
- Dieser Speicherbereich endet bei Adresse 40959.

Dem Programmierer stehen 38911 Speicherplätze zur Verfügung. Alle diese Speicherzellen sind also RAM-Zellen, das heißt, man kann Daten hineinschreiben und herauslesen.

- Von 40960 bis 49151 sind die fest vorgegebenen Programme zum Übersetzen von Basic in den Maschinencode enthalten
- Anschließend an 53248 bis 57343 sind alle Buchstaben und Zeichen und die Ein- und Ausgabeprogramme gespeichert.
- Danach folgen die Programme des Betriebssystems, auch Kernel genannt.

Bemerkenswert ist, daß die Betriebsprogramme des Computers (Basic-Übersetzer, Zeichensatz, Ein- und Ausgabe und Kernel) in einem ROM-Speicher untergebracht sind, der über einem RAM-Speicher sitzt. Das heißt sie haben dieselben Adressen. Wie dieser RAM-Teil des Speichers genutzt werden kann, gehört zur höheren Programmierkunst und wird hier nicht behandelt.

Uns stellt sich als nächstes die Frage, wie wir Daten in einzelne Speicherzellen des Programmspeichers hineinschreiben und herauslesen können.

21.2. Im Speicher stöbern

Basic hat zwei Befehle, mit deren Hilfe der Speicher abgefragt werden kann.

Während der Lese-Befehl völlig ungefährlich ist - er »kopiert« praktisch den Inhalt der Speicherzelle und läßt das Original unverändert - kann der Schreib-Befehl gefährlich sein. Ich will Sie gleich zu Beginn davor warnen, ihn unbedacht einzusetzen - er verändert den Inhalt einer Speicherzelle, und über die Folgen dieser Veränderung für ein Pro-

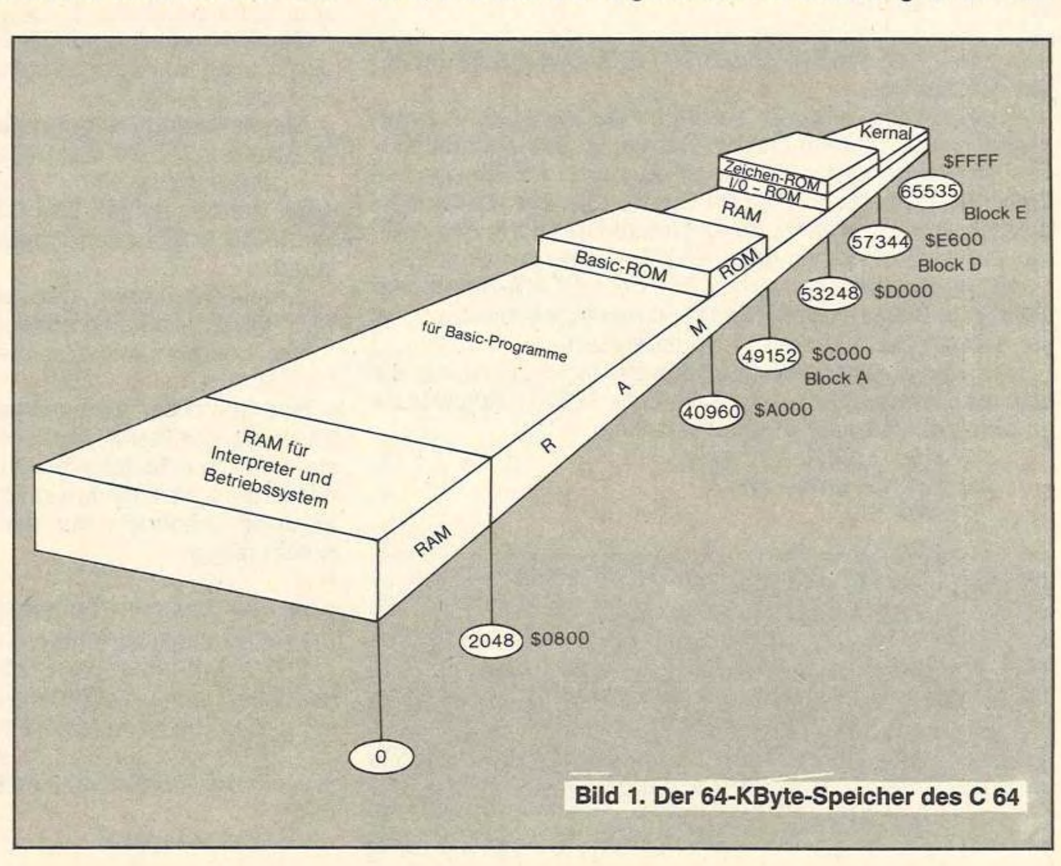


Bild 1. Der 64-KByte-Speicher des C 64



Basic-Befehl Nr. 27 POKE
- er wird in der folgenden Weise geschrieben:
  POKE Adresse,Wert
Adresse und Wert müssen durch ein Komma voneinander getrennt sein
- der POKE-Befehl speichert den »Wert« in der mit »Adresse« bezeichneten Speicherzelle ab. Er überschreibt dabei einen dort gespeicherten früheren Wert
- gültige »Werte« liegen im Bereich von minimal 0 bis maximal 255. Wird dieser Bereich überschritten, erscheint die Fehlermeldung ILLEGAL QUANTITY ERROR.
- der erlaubte Bereich für »Adresse« reicht von 0 bis 65535. Ein Überschreiten wird mit der Fehlermeldung ILLEGAL QUANTITY ERROR bestraft.

gramm oder für den Betrieb des Computers muß man sich im klaren sein. Das heißt freilich nicht, daß Ihrem C 64 dadurch ernstlich Gefahr drohen könnte. Im schlimmsten Fall schalten Sie den Computer einfach aus und wieder ein – danach ist alles wie vorher.
Der Schreib-Befehl heißt

POKE

was das englische Wort für »hineinstochern« ist (nomen est omen). Nach dem Befehlswort muß die Adresse der Speicherzelle stehen, danach folgt, durch ein Komma getrennt, die Zahl, die hineingeschrieben werden soll.
Der Befehl

```
POKE 14000,55
```

schreibt also die Zahl 55 in die Speicherzelle 14000.
Der Lese-Befehl heißt

PEEK

das ist das englische Wort für »hineinschauen«. Hinter diesem Befehlswort steht die Adresse – und zwar in Klammern! – deren Inhalt gelesen werden soll.
Die Befehlsfolge:

```
A=PEEK(14000):PRINT A
```

oder noch kürzer

```
PRINT PEEK(14000)
```

liest den Inhalt der Speicherzelle 14000 und druckt ihn auf den Bildschirm.

In der ersten Zeile wird der Inhalt der Variablen A zugeordnet und steht unter diesem Namen im Speicher für spätere Verwendung. Da der Inhalt aber trotz PEEKen in der Zelle 14000 stehenbleibt und von dort immer wieder herausgeholt werden kann, bietet sich die Kurzform der zweiten Zeile an, die direkt das PEEK-Ergebnis ausdruckt.

Wir haben gerade vorhin mit dem POKE-Befehl die Zahl 55 in die Speicherzelle 14000 hineingeschrieben und sie danach mit PEEK wieder ausgelesen.

Nun, das beweist natürlich noch gar nichts. Schauen wir also mal im Speicher rund um die Zelle 14000 nach, was da so drin steht. Mit den Programmzeilen:

```
10 FOR I=13900 TO 14200
20 PRINT I,PEEK(I)
30 NEXT I
```

Basic-Befehl Nr. 28 PEEK
- er wird in der folgenden Weise geschrieben:
  PEEK (Adresse)
- die »Adresse« muß immer in Klammern stehen
- der Befehl liest den Inhalt der durch die »Adresse« angegebenen Speicherzelle
- der erlaubte Bereich von »Adresse« reicht von 0 bis 65635. Wird er überschritten, meldet dies der Computer mit ILLEGAL QUANTITY ERROR

schauen wir uns den Speicherbereich von 13900 bis 14200 an. Es wird durch Zeile 20 der jeweilige Wert von I und daneben die Zahl, die in der Adresse I gespeichert ist, mit nur einem PRINT-Befehl ausgedruckt. Das Komma zwischen den beiden bewirkt einen Abstand von einer Viertelzeilenlänge.

Nach RUN sehen wir, daß in allen Speicherzellen entweder eine 0 oder 255 steht, mit Ausnahme der Zelle 14000, da steht brav unsere 55.

Übrigens hoffe ich, Sie wissen, daß mit der CTRL-Taste links oben der Ablauf des Programms gebremst werden kann – zum besseren Überblick, wenn die Speicherzelle 14000 vorbeisaust. In diesem Teil des Speichers steht also praktisch gar nichts. Das ist auch kein Wunder, denn wir tummeln uns ja im oberen Teil des RAM-Speichers, der uns für Basic-Programme zur Verfügung steht. Unser Mini-Programm von drei Zeilen, welches ab der Speicherzelle 2048 gespeichert wird, reicht da natürlich bei weitem nicht hin. Wenn Sie Zeile 10 so abändern, daß der Ausdruck ab dem Speicheranfang 2048 beginnt:

```
10 FOR I=2048 TO 2200
```

dann sehen wir in der Tat ein Durcheinander von Zahlen, die bis hin zur Adresse 2095 reichen. Das ist – in einem besonderen Code geschrieben – unser Programm. Danach kommen wieder die Leerserien mit 0 und 255.

Schauen wir spaßeshalber noch in den obersten Teil des Speichers, wo laut Speicherbild die mysteriösen Register für Ein- und Ausgabe liegen. Ich wähle Speicherzelle 53265 als Versuchskaninchen.

```
PRINT PEEK (53265)
```

Das ergibt eine 155.

Jetzt machen wir ein Experiment – das, wie gesagt, unerwünschte Folgen haben kann. Wir ändern mutwillig den Inhalt dieser Speicherzelle mit:

```
POKE 53265,16
```

Und siehe da, nach Drücken der RETURN-Taste verschiebt sich der obere Rand des Bildschirms und schneidet alle Zeichen in der ersten Zeile ab. Sie sind zwar noch da, aber nicht sichtbar.

Der Originalzustand läßt sich mit dem oben ermittelten »Normalwert« wiederherstellen, indem Sie eintippen:

```
POKE 53265,155
```

Dieser Versuch ist gutgegangen. Aber wenn Sie statt der 15 oder der 155 die Zahl 33 in die Zelle POKEn:

```
POKE 53265,33
```

dann geht es schief. Der Bildschirm wird leer und durch nichts läßt er sich wiederbeleben – der Computer ist »abgestürzt«!

Ein Mittel bleibt uns doch, nämlich die <STOP RESTORE>-Tastenkombination.

Hier noch ein anderes interessantes Beispiel:

```
POKE 649, 0
```

Der Computer nimmt danach keine Eingabe mehr an, das heißt, der Tastaturpuffer (Normalwert 10) ist auf null gestellt, und die Tastatur blockiert. Dies könnte innerhalb eines Programmablaufs sinnvoll sein. Im Direkt-Modus »rettet« uns wiederum nur die <RUN/STOP RESTORE>-Kombination.

21.3. Wir POKEn noch ein Weilchen

Es gibt keine Computerzeitschrift, die nicht unter der Rubrik Tips und Tricks alle möglichen POKE-Adressen angibt, mit denen sich verblüffende Effekte erzielen lassen. Auch ich gebe Ihnen ein paar Hinweise.

```
POKE 199,1:PRINT "ABCDE"
```

druckt alle Zeichen dieser Programmzeile revers (invertiert).

```
POKE 650,64
```

schaltet die Wiederholfunktion aller Tasten aus

```
POKE 650,0
```

nur die Leer-, INST/DEL- und alle Cursor-Tasten wiederholen, solange sie gedrückt werden

```
POKE 650,128
```

alle Tasten haben Wiederholfunktion (Normalzustand)

```
POKE 53281,4
```

schaltet die Farbe des Bildschirm-Hintergrundes auf Violett

```
POKE 53280,5
```

schaltet die Farbe des Bildschirm-Rahmens auf Grün.

Die beiden letzten POKE-Adressen 53281 und 53280 wollen wir uns näher anschauen.

Der Zahlenwert in der Speicherzelle 53281 bestimmt also die Hintergrundfarbe, während in Speicherzelle 53280 die Rahmenfarbe festgelegt ist. Da nach einem POKE-Befehl Zahlen von 0 bis 255 zugelassen sind, ist es sicher ganz interessant, welche Zahl welche Farbe hervorruft. Ein kleines Programm gibt uns darüber Auskunft:

```
10 FOR I=0 TO 255
20 POKE 53281,I
30 PRINT I
40 GET A$:IF A$="" THEN 40
50 NEXT I
```

Zwischen den Zeilen 10 und 50 wird in einer Schleife die Variable I von 0 bis 255 hochgezählt.

Der jeweilige Wert von I wird in Zeile 20 in die Speicherzelle 53281 gePOKEt und ändert dadurch die Hintergrundfarbe.

Merken wir uns:
1. Der POKE-Befehl ist nützlich und gefährlich zugleich.
2. Wird eine bestimmte Zahl in eine Speicherzelle gePOKEt, mit der der Computer seine eigenen Abläufe steuert (Bereiche 0 bis 2047, 40960 bis 49152 und 53248 bis 65535), kann dadurch der Ablauf beeinflußt werden. Es kann aber auch zum »Absturz« des Computers kommen.
3. Mit »Absturz« wird beim Computer der Zustand bezeichnet, in dem kein Programm mehr läuft und der Computer auf keine normalen Steuertasten mehr reagiert.
4. Ein abgestürzter Computer kann nur durch Aus- und Wiedereinschalten wieder in Gang gesetzt werden. Nach dem Aus- und Einschalten befindet sich der Computer im Anfangszustand, das heißt, ein Programm, das vorher im Arbeitsspeicher war, ist verloren.
5. Es ist empfehlenswert, bevor ein POKE-Befehl ausgeführt wird, sei es im Direkt-Modus oder sei es innerhalb eines Programms, ein im Arbeitsspeicher befindliches Programm zuerst auf Band oder Diskette abzuspeichern – für alle Fälle !

Um die Zugehörigkeit der Farben zu den Zahlen zu sehen, wird in Zeile 30 der jeweilige Wert von I ausgedruckt.

Zeile 40 dient dazu, die Schleife schrittweise weiterzuschalten. Der GET-Befehl in dieser Zeile springt so lange auf seine eigene Zeilennummer zurück, bis irgendeine beliebige Taste gedrückt wird. Erst dann kommt der NEXT-Befehl in Zeile 50 zum Zuge.

Mit diesem kleinen Programm werden pro Tastendruck alle Farben durchgeleiert, wobei die jeweils unterste ausgedruckte Zahl dem Farbwert entspricht.

Was wir vorher mit der Speicherzelle 53281 für die Hintergrundfarbe gemacht haben, können wir ebenso mit der Zelle 53280 für die Umrandung machen. Diese beiden Adressen gehören zu den »Registern« des VIC-Bausteins, der für Töne und Grafik zuständig ist.

Merken wir uns:
1. Ein REGISTER ist eine Speicherzelle im Mikroprozessor (CPU) oder in einem anderen elektronischen Baustein. Im C64 besteht ein Register aus 8 Bit, das ist 1 Byte.
2. In einem Register werden Daten gespeichert, welche den Ablauf von arithmetischen, logischen oder von Steueroperationen festlegen.
3. Der Inhalt von Registern kann mit PEEK ausgelesen und, was viel wichtiger ist, mit POKE verändert werden.

Jetzt kennen wir also die beiden Farbregister und wissen, wie wir die Farben des Bildschirms unseren Wünschen anpassen können.

22 Der Bildschirmspeicher und Farbspeicher

Ich möchte Sie gern noch ein bißchen länger mit dem Speicher beschäftigen. Schauen Sie sich bitte noch mal das Speicherbild aus Teil 21 an. Da sehen wir ab Adresse 1024 das »Bildschirm-RAM«.

Um zu demonstrieren, was das ist, zeige ich Ihnen ein kleines Experiment:
- Löschen Sie den Bildschirm mit der CLEAR-Taste
- Fahren Sie mit dem Cursor in die untere Hälfte des Bildschirms.
- Geben Sie direkt ein:

```
POKE 1024,1
```

Ganz links oben auf dem Bildschirm steht plötzlich ein A. Wenn Sie aus der 1 eine 2 machen und den Befehl wiederholen, verwandelt sich das A in ein B.

Und noch ein Versuch: ändern Sie 1024,2 in 1025,3 um und geben es ein. Jetzt steht neben dem B ein C.

Die Zahl 1 erzeugt also ein A, 2 ein B und 3 ein C. Dann müßte eigentlich die 26 ein Z hervorrufen – was sie auch tut.

Wir haben also einen neuen Code für die Zeichen auf dem Bildschirm gefunden.

Ein ähnlicher Zusammenhang deutet sich an bei den Adressen: 1024 setzt den Buchstaben an den ersten Platz des Bildschirmes, 1025 an den zweiten. Der Bildschirm hat 40 Stellen pro Zeile und das für 25 Zeilen. Das macht insgesamt 1000 Plätze auf dem Bildschirm.

Daher müßte die Adresse 2023 einen Buchstaben ganz rechts unten plazieren.

```
POKE 2023,26
```

setzt ein Z genau dorthin, wie vorhergesagt.

Ich will Sie nicht länger plagen und das alles zusammenfassen:

Merken wir uns:
1. Der Elektronenstrahl, der mit großer Geschwindigkeit über den Bildschirm des Fernsehers oder des Monitors flitzt und dort Bilder oder Text hinmalt, hat kein Gedächtnis. Deswegen muß der Computer alle Angaben, die der Elektronenstrahl braucht, in einem gesonderten Speicher festhalten, der deshalb »Bildschirmspeicher« heißt.
2. Im Bildschirmspeicher sind alle Zeichen gespeichert, die zum jeweiligen Zeitpunkt auf dem Bildschirm erscheinen.
3. Für den Bildschirmspeicher sind im RAM die Speicherzellen 1024 bis 2047 reseviert. Da auf dem Bildschirm des C64 genau 1000 Plätze vorhanden sind (40 Stellen ►



mal 25 Zeilen), reicht der Bildschirmspeicher nur bis Adresse 2123. Die restlichen 24 Byte sind frei.

4. Alle Zeichen und Buchstaben stehen im Bildschirmspeicher mit einem speziellen Code, der im Bedienungshandbuch auf den Seiten 133 und 134 aufgelistet und auf Seite 132 erklärt ist (C 128-Handbuch: Anhang A-4 bis A-6). Der Bildschirmcode hat nichts mit dem ASCII-Code zu tun.

Durch direktes POKEn von Bildschirm-Codewerten in den Bildschirmspeicher lassen sich Effekte erzielen, die mit dem PRINT-Befehl nur sehr umständlich möglich wären. Ich will Ihnen das an einem Beispiel zeigen.

Ziel des Experiments soll es sein, eine bewegte farbige Umrandung des Bildschirms zu erzeugen, die als Programmteil in anderen Programmen verwendbar ist. Ich verwende dazu ein Beispiel aus dem Buch »VC 20 Spielebuch« von A.Dripke, das viele gute Ideen und Anleitungen enthält, die sich auch für die Arbeit mit dem C 64 nutzbringend verwenden lassen. Wie üblich gehe ich in Stufen vor.

```
→ 10 FOR I=0 TO 999
  20 POKE 1024+I,42
  30 NEXT
```

Diese drei Zeilen zählen vom Anfang des Bildschirmspeichers (1024) 1000 Plätze hoch (von 0 bis 999) und POKEn in jeden Platz einen Stern. Der Bildschirmcode des Sternes ist 42 (siehe oben erwähnte Tabelle).

Auffallend bei diesem eindrucksvollen Vorgang ist, daß der Cursor und die READY-Meldung nicht anschließend an den letzten Stern erscheint, sondern dort, wo der letzte Basic-Befehl – in unserem Fall das RUN – auf den Bildschirm geschrieben worden ist.

Im nächsten Schritt lassen wir den Stern nur auf der obersten und untersten Zeile laufen. Die Schleife zählt daher jetzt nur bis 39 (Zeilenlänge):

```
→ 10 FOR I=0 TO 39
  20 POKE 1024+I,42
  30 POKE 1024+960+I,42
  40 NEXT I
```

Neu ist hier die Zeile 30. Sie erhöht die Adresse um 24*40=960 Plätze und beginnt dadurch in der untersten Zeile.

Dieses Programm malt also eine Rahmenlinie oben und unten. Jetzt fehlt noch links und rechts.

```
→ 50 FOR K=0 TO 960 STEP 40
  60 POKE 1024+K,42
  70 POKE 1024+39+K,42
  80 NEXT K
  90 GOTO 10
```

Um von oben nach unten zu zählen, beginnen wir mit 0, gehen aber in 40er-Schritten gleich an den Anfang der jeweils nächsten Zeile bis zum Anfang der letzten Zeile. Das gibt uns in Zeile 60 den linken Rand.

Für die andere Seite benutzt die Zeile 70 die gleiche Zählung, POKEt aber den Stern um 39 Plätze verschoben, also am rechten Rand. Ganz zum Schluß springen wir in Zeile 90 auf den Anfang zurück, um das lästige READY zu verhindern.

Jetzt wollen wir diese Umrandung bunt machen, und zwar nach jedem Umlauf in einer anderen Farbe.

Auch hier bietet der Computer eine Lösung an. Zwischen den Adressen 55296 und 56319 liegt das »Farb-RAM«.

Dieser Bereich im RAM ist der Zwilling zum Bildschirmspeicher, nur ist er für die Farben zuständig.

Wenn wir jetzt das wiederholen, was wir gleich nach der Überschrift »Bildschirmspeicher« gemacht haben und POKEn zusätzlich in die erste Zelle des Farbspeichers 2048 die Zahl 100, dann ändern wir die Farbe des Zeichens, also:

```
→ - Bildschirm löschen
  - Cursor in die untere Hälfte
  - POKE 1024,1
  - POKE 55296,7
```

Das A erscheint in Gelb.

Insgesamt haben wir 16 Farben zur Verfügung. Sie entsprechen den folgenden Werten:

0 = schwarz	8 = orange
1 = weiß	9 = braun
2 = rot	10 = hellrot
3 = lila	11 = dunkelgrau
4 = purpur	12 = mittelgrau
5 = grün	13 = hellgrün
6 = blau	14 = hellblau
7 = gelb	15 = hellgrau

Merken wir uns:

1. Im Farbspeicher sind alle Farben gespeichert, in welchem die Zeichen auf dem Bildschirm erscheinen.
2. Für den Farbspeicher sind im RAM die Speicherzellen 55296 bis 56319 reserviert.
   Da auf dem Bildschirm des C64 genau 1000 Plätze vorhanden sind (40 Stellen mal 25 Zeilen), reicht der Farbspeicher nur bis Adresse 55295. Die restlichen 24 Byte sind frei.
3. Als Codewerte für die Farben gelten die Zahlen von 0 bis 15.

Wir wollten aber eigentlich das Umrahmungs-Programm mit Farben versehen. Dies erfolgt mit POKE in den Farbspeicher.

Ich habe folgendes gemacht (Listing 6):

Die Zeilen 5 und 90 bilden die übergeordnete Schleife, innerhalb der wir die Farbe F in den Farbspeicher POKEn.

Um sicherzustellen, daß die Farbe F in diejenigen Speicherzellen des Farbspeichers kommt, die denen des Bildschirmspeichers entsprechen, habe ich an jeden Bildschirm-POKE-Befehl ein POKE 55296 angehängt mit denselben Argumenten. Nur hinter dem Komma steht ein F für die Farbe und nicht die 42 für den Stern.

Sie sehen, die Arbeit des Ausrechnens der Adresse braucht man nur einmal zu machen.

Jetzt habe ich noch eine Variante vor:

Die Umrandung soll nicht oben und unten beziehungsweise links und rechts gleichzeitig laufen, sondern immer im Kreis.

Bisher hatten wir diese Situation:

Zeile 20: oben, von links nach rechts
Zeile 30: unten, von links nach rechts
Zeile 60: links, von oben nach unten
Zeile 70: rechts, von oben nach unten.

```
5 FOR F=0 TO 15
10  FOR I=0 TO 39
20   POKE 1024+I,42:POKE 55296+I,F
30   POKE 1024+960+I,42:POKE 55296+960+I,F
40  NEXT I
50  FOR K=0 TO 960 STEP 40
60   POKE 1024+K,42:POKE 55296+K,F
70   POKE 1024+39+K,42:POKE 55296+39+K,F
80  NEXT K
90 NEXT F
```

Listing 6. Farbige Umrandung

Wir müssen das so abändern:
Zeile 20: oben, bleibt
Zeile 70: rechts, bleibt
Zeile 30: unten, von rechts nach links
Zeile 60: links, von unten nach oben

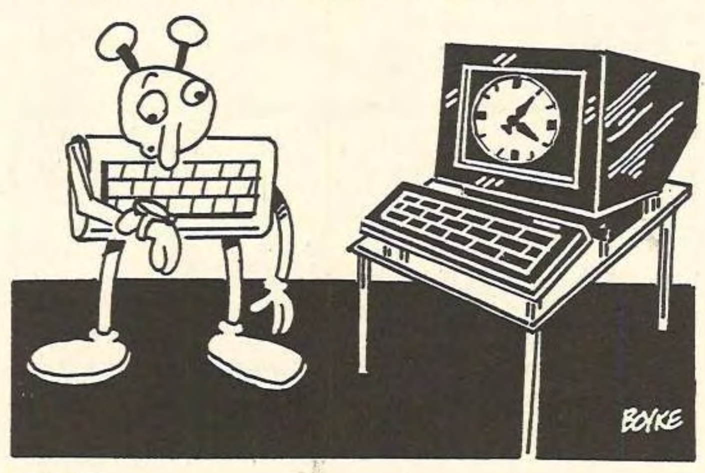
Die Reihenfolge ändert sich also, und die Zeilen 30 und 60 laufen in der entgegengesetzten Richtung. Um das zu erreichen, müssen wir leider für jede POKE-Zeile eine eigene Schleife bauen. In den Zeilen 30 und 60 wird rückwärts gezählt, mit negativem STEP.

In der neuen Reihenfolge sieht das so aus (Vorsicht, neue Zeilennummern!):

```
5 FOR F=0 TO 15
10  FOR I=0 TO 39
20   POKE 1024+I,42:POKE 55296+I,F
30  NEXT I
40  FOR K=0 TO 960 STEP 40
50   POKE 1024+39+K,42:POKE 55296+39+K,F
60  NEXT K
70  FOR I=39 TO 0 STEP -1
80   POKE 1024+960+I,42:POKE 55296+960+I,F
90  NEXT I
100  FOR K=960 TO 0 STEP -40
110   POKE 1024+K,42:POKE 55296+K,F
120  NEXT K
135 NEXT F
140 GOTO 5
```

Listing 7. Umlaufende Umrandung

Spätestens jetzt könnten Sie sich wieder einmal eine Pause gönnen, denn in den nächsten Lektionen wollen wir uns einige neue Bereiche des C 64 erarbeiten.



23 Die Uhr des Computers

Der C 64 hat eine innere Uhr eingebaut, deren Stand abgefragt, ausgedruckt und somit zu Messungen und zur Programmsteuerung eingesetzt werden kann. Diese Uhr startet beim Einschalten mit dem Stand 0 und läuft, bis sie durch einen entsprechenden Befehl auf Null oder auf irgendeinen anderen Wert gesetzt wird. Der aktuelle Stand dieser Uhr kann mit der dafür fest reservierten Variablen

TI

abgefragt werden. Mit der ewigen Schleife:

```
10 PRINT TI:GOTO 10
```

drucken wir ein laufendes Band von sich schnell ändernden Zahlen auf den Bildschirm.

Diese Zahl durch 60 geteilt gibt uns die Zeit seit dem Loslaufen (Einschalten) in Sekunden an, durch 3600 geteilt in Minuten und durch 216000 geteilt in Stunden.

Weil diese Darstellung etwas mühsam ist, wenn man eine echte Zeitangabe braucht, besitzt Basic noch eine andere reservierte Variable

TI$

welche die Zeit in einer sechsstelligen Zahl ausdruckt. Dabei bedeutet 124533 12 Stunden, 45 Minuten und 33 Sekunden. Dies testen wir mit.

```
10 PRINT TI$:GOTO 10
```

Dieses Zahlenband verändert sich jetzt im Rhythmus von Sekunden.

TI$ ist auch die Variable, mit dem die Uhr auf einen beliebigen Wert - auch auf Null - gesetzt wird:

```
TI$="000000"
```

setzt die Uhr auf Null.

```
TI$="221500"
```

setzt die Uhr auf 22 Uhr 15.

In dem Augenblick, wenn Sie nach diesen Befehlen die RETURN-Taste drücken, läuft die Uhr mit diesem neuen Anfangswert los.

Basic-Variable TI, TI$

- Beim Einschalten des Computers läuft ein interner Zähler los, der 60mal in jeder Sekunde um 1 erhöht wird. Wenn der Zähler den Stand 5184000 erreicht hat - das entspricht einer Laufzeit von 24 Stunden, wird er auf 0 zurückgesetzt.
- Mit TI kann der Stand des Zählers abgefragt werden.
- Die Variable TI$ wandelt den Zahlenwert des Zählers in eine sechsstellige Zahl um, deren erste beide Stellen die Stunden der Uhrzeit, die mittleren beiden die Minuten und die letzten beiden die Sekunden angeben.
- Mit der Anweisung TI$= "aabbcc" wird die interne Uhr auf aa Uhr bb Minuten cc Sekunden gestellt.

Mit einer Simulation einer Stoppuhr wollen wir die bei den Variablen TI und TI$ in einem Programm anwenden. Mit der Stoppuhr wollen wir den Vergleich der Laufzeiten der Zählschleife mit IF-THEN und der FOR-TO-NEXT-Schleife aus Lektion 8 wiederholen.

23.1. Stoppuhr

Unsere Stoppuhr läuft in dem Moment los, wo sie auf Null gesetzt wird:

```
10 TI="000000"
```

Sie stoppt mit der letzten Zeile der Messung:

```
90 PRINT TI:END
```

Um die Meßzeit in Sekunden zu erhalten, wenden wir noch obige Regel an, teilen durch 60 und drucken aus, was wir sehen werden. Zeile 90 wird verbessert zu:

```
90 PRINT TI/60 "SEKUNDEN"
```

Zwischen 10 und 90 plazieren wir unser Testprogramm. Ich schlage vor, genau wie in Lektion 8 den Bildschirm mit 375 A zu füllen.

Die ungleichen Lücken zwischen den Zeilen, sowohl in der Numerierung als auch in der Schreibweise, dienen nur der Lesbarkeit, haben aber auf den Ablauf keinen Einfluß.

Die 80er Zeilen rücken den Ausdruck des Ergebnisses nach unten.

Ergebnis:
ZÄHLER+GOTO-Schleife: 3,9333 s
FOR-NEXT-Schleife: 0,9666 s

Die FOR-NEXT-Schleife ist strahlender Sieger !!

Diese Technik wollen wir verwenden, um eine Zeitbegrenzung so in ein Programm einzubauen, daß es nach ei-



Testprogramm »Stoppuhr«	
ZÄHLER+GOTO	FOR-TO-NEXT
10 TI$="000000"	110 TI$="000000"
20 PRINT CHR$(147)	120 PRINT CHR$(147)
	130 FOR X=0 TO 374
40 PRINT "A";	140 PRINT "A";
50 IF X=374 THEN 80	
60 X=X+1	
70 GOTO 40	170 NEXT
80 PRINT	180 PRINT
90 PRINT TI/60"SEK"	190 PRINT TI/60"SEK"
99 END	199 END

ner gewissen Laufzeit abgebrochen, beziehungsweise beendet wird.

23.2. Zeitabhängige Programmunterbrechung

Wenn das Programm im Prinzip aus einer oder mehreren Schleifen besteht, ist die Lösung einfach. Ist es ein gerade verlaufendes Programm, das an irgendeiner Stelle unterbrochen werden soll, wird die Sache schon schwieriger, weil die Abfrage der Zeit eigentlich dauernd erfolgen muß.

Wir nehmen uns hier den leichteren Fall vor. Sie kennen doch sicher das alte Spiel »Stadt, Land, Fluß«, bei dem innerhalb einer festgesetzten Zeit aus jedem Sachgebiet ein Beispiel mit denselben Anfangsbuchstaben aufzuschreiben ist. Eine Abwandlung dieses Spiels wähle ich als Anwendung einer Zeitsteuerung, die ich – wie immer – schrittweise mit Ihnen entwickeln möchte.

1. Initialisierung

Mit diesem Fremdwort bezeichnen wir das Herstellen des Anfangszustandes eines Programms. In unserem Fall soll festgelegt werden:
- ein Sachgebiet (S$)
- die Zeitdauer (Z)
- der Anfangsbuchstabe (B$)

Sachgebiet und Zeitdauer werden vom Spieler per INPUT eingegeben, der Buchstabe (von A bis Z) wird mit einem Zufallsgenerator erzeugt.

```
100 PRINT CHR$(147)
110 INPUT "WAEHLE EIN SACHGEBIET";S$
120 INPUT "STELLE DIE UHR AUF";Z
130 B$=CHR$(INT(RND(0)*26+65))
```

Ich glaube, zu den Zeilen 100 bis 120 ist nichts weiter zu sagen.

Zeile 130 soll eine Zufallszahl erzeugen, deren ASCII-Code laut Tabelle auf Seite 135 des Bedienungshandbuches zwischen 65 (A) und 90 (Z) liegt. Das Kochrezept dazu habe ich in Lektion 10 erklärt. Seine Formel lautet wie folgt:

Ganze Zahlen innerhalb des Zahlenbereichs von minimal Y bis maximal (Y+X-1) werden erzeugt durch:

INT(RND(0)*X+Y)

Zeile 130 wendet diese Formel an. Y liegt mit 65 fest, X errechnet sich aus (Y+X-1)=95 und ergibt 26. Bei Zeile 130 bitte auf die Anzahl der Klammern aufpassen!

Zeile 140 faßt die Initialbedingungen zusammen. Sie ist ein Beispiel für das Einfügen von Wörtern in einen vorgegebenen Text. Wichtig sind dabei die Leerzeichen im Text vor und nach den Stringvariablen S$ und B$. Versuchen Sie es ohne Leerzeichen, und Sie werden das unlesbare Resultat sehen.

Zeilen 230, 240 und 250 geben den Startschuß.

```
→ 140 PRINT"NENNE "S$", DIE MIT "B$" ANFANGEN"
  230 PRINT"START MIT IRGENDEINER TASTE":
  240 GET X$:IF X$=""THEN 240
  250 TI$="000000"
```

Jetzt läuft eine Zeitschleife los, die wie im letzten »Weckprogramm« einen vorgegebenen Zeitwert Z mit dem Stand der eingebauten Uhr TI$ vergleicht.

```
→ 270        (Wörter eingeben)
  340 IF VAL(TI$) <> Z THEN 270
```

Innerhalb der Zeilen 270 und 340 soll folgendes passieren:
- Wörter W$ eingeben
- Anfangsbuchstaben B$ überprüfen
- Wörter ausdrucken
- richtige Wörter zählen (R)

Zum Eingeben von Wörtern steht uns INPUT und GET zur Verfügung.

INPUT erlaubt die Eingabe von kompletten Wörtern, indem es auf einzelne Zeichen wartet – macht aber damit die Wirkung der Zeitmeß-Schleife zunichte.

GET wartet nicht und ermöglicht der Zeitschleife, in den Ablauf einzugreifen – nimmt aber nur einzelne Zeichen entgegen. Die Wirkung der Zeitschleife ist mir wichtiger, zumal man aus einzelnen Zeichen auch Wörter zusammensetzen kann! Wir nehmen also GET (Zeile 270).

```
100 PRINT CHR$(147)                              <129>
110 INPUT"WAEHLE EIN SACHGEBIET";S$              <235>
120 INPUT"STELLE DIE UHR AUF";Z                  <082>
130 B$=CHR$(INT(RND(0)*26+65))                   <006>
140 PRINT"NENNE "S$", DIE MIT " B$ " ANFAN
    GEN                                          <010>
200 :                                            <176>
210 REM**** ZEITSCHLEIFE & EINGABE ****          <029>
220 :                                            <196>
230 PRINT"START MIT IRGENDEINER TASTE"           <158>
240 GET X$:IF X$=""THEN 240                      <214>
250 TI$="000000"                                 <229>
260 :                                            <238>
270 GET E$                                       <002>
280 W$=W$+E$                                     <152>
290 IF E$<>CHR$(13) THEN 340                     <039>
300 IF LEFT$(W$,1)<>B$ THEN 330                  <200>
310 PRINT W$;                                    <001>
320 R=R+1                                        <230>
330 W$=""                                        <193>
340 IF VAL(TI$)<>Z THEN 270                      <239>
400 :                                            <122>
410 REM**** STOP & ERGEBNIS ******               <091>
420 :                                            <142>
430 FOR F=0 TO 14                                <014>
440 POKE 53280,F:NEXT F                          <120>
450 PRINT R                                      <188>
```


Listing 8. Die eingebaute Uhr

Das Zusammensetzen eines Wortes besorgt uns Zeile 280, indem die einzelnen Buchstaben E$ zu einem Wort W$ einfach dazuaddiert werden:

```
→ 270 GET E$
  280 W$=W$+E$
```

Mit den Zeilen 270, 280 und 340 haben wir eine Schleife, die solange Wörter zusammensetzen würde, bis der Zeitvergleich alles abbricht. Versuchen Sie es mit RUN 270!

Ein Wort wird dadurch abgeschlossen – wie bei INPUT – daß die RETURN-Taste gedrückt wird. Dann allerdings soll das Wort ausgedruckt und als Resultat (R) gezählt werden – wenn es richtig ist. Die primitivste Prüfung ist die des Anfangsbuchstabens.

```
→ 290 IF E$ <> CHR$(13) THEN 340
```

Zeile 290 prüft auf Drücken der RETURN-Taste, die den ASCII-Code 13 hat. Das heißt, sie prüft, ob die Taste nicht gedrückt ist. Dann rückt das Programm auf Zeile 340 weiter, mißt die Zeit, springt auf Zeile 270 zurück und schaut nach, ob ein neuer Buchstabe E$ eingegeben worden ist.

Ist <RETURN> aber gedrückt, geht das Programm mit der Zeile 300 weiter. Diese schneidet sich mit dem

LEFT$-Befehl den ersten Buchstaben des Wortes W$ ab und vergleicht ihn mit dem vorgegebenen Anfangsbuchstaben B$.

```
→ 300 IF LEFT$(W$,1) <> B$ THEN 330
  310 PRINT W$
  320 R=R+1
  330 W$=""
```

Ist die Prüfung auf »ungleich« nicht erfüllt, das heißt sind beide Buchstaben gleich, dann druckt Zeile 310 das Wort W$ aus und zählt es als richtiges Resultat R zu eventuell schon vorhandenen richtigen Resultaten dazu. Danach wird das eingegebene Wort W$ in Zeile 330 wieder gelöscht.

Ergibt der Vergleich der beiden Buchstaben in Zeile 300 aber, daß sie ungleich sind, dann überspringt Zeile 300 diesen ganzen Teil, löscht in Zeile 330 das bisher eingegebene (aber falsche) Wort und macht in Zeile 340 mit der Schleife weiter.

Nach Ablauf der Schleife, oder besser gesagt, nach Abbruch durch die Uhr, läßt Zeile 430/440 den Bildschirmrahmen bunt aufflimmern und Zeile 450 druckt das Resultat R, nämlich die Anzahl der eingegebenen Wörter aus:

```
430 FOR F=0 TO 14
440 POKE 53280,F:NEXT F
450 PRINT R
```

In Listing 8 ist das bisherige Programm vollständig dargestellt. Ich habe es dabei mit REM-Zeilen in funktionelle Blöcke unterteilt.

24 Die Technik der Unterprogramme

Das Spiel »Stadt, Land, Fluß« wird immer mit mehreren Personen gespielt. Obwohl mir klar ist, daß ein derartiges Vorhaben mit dem Computer nicht ganz einfach zu organisieren ist – kein Spieler soll die Wörter des vorhergehenden Spielers sehen können – lasse ich dieses Detail außer acht und zeige Ihnen, wie man so einen mehrfachen Ablauf desselben Programms programmiert.

Wir brauchen dazu folgende Schritte:
- Initialisierung, wie vorher
- Frage nach der Zahl der Mitspieler
- Schleife für entsprechend viele Durchgänge
- Einzeldurchgänge pro Spieler
- Speicherung der Einzelergebnisse
- Ausdruck des Gesamtergebnisses

Die Initialisierung mit den Zeilen 100 bis 130 bleibt also gleich.

Jetzt stehen wir vor dem Problem, den eigentlichen Spielteil von Zeile 300 bis Zeile 340 so oft laufen zu lassen, wie Mitspieler vorhanden sind. Völlig unsinnig wäre es, diesen Programmteil zu vervielfachen, da ja die Anzahl der Mitspieler variabel sein soll.

Also müssen wir den Spielteil, der durch die Zeilen 200 bis 340 gebildet wird, bei jedem Spieldurchgang neu »anspringen«. Dazu legen wir ihn zuerst einmal an das Ende des Programms. Das geht am schnellsten dadurch, daß Sie vor jede Zeilennummer eine 1 einfügen, wodurch wir einen Programmblock von 1200 bis 1340 erhalten, die Abstands- und REM-Zeilen mit eingeschlossen.

Vorsicht! Wir haben innerhalb dieser Zeilen zwei Sprungadressen, die wir ebenfalls ändern müssen: in Zeile 290 und in Zeile 300.

Die »alten« Zeilen 200 bis 340 müssen wir durch Eintippen ohne Inhalt der Reihe nach löschen.

Wenn Sie das Programm LISTen, dann sitzt in der Tat der eigentliche Spielteil jetzt am Ende des Programms, und Zeilen 200 bis 340 sind verschwunden.

Dasselbe machen wir mit dem anderen Programm-Teilchen »STOP & ERGEBNIS«, das wir ebenfalls öfters verwenden wollen. Wir schieben es mit der oben genannten Methode auf 1400 bis 1450 und löschen Zeilen 400 bis 450.

Jetzt haben wir Platz, um die Frage nach der Anzahl der Mitspieler einzubauen. Ich schiebe die Frage noch vor die Spielanweisung in Zeile 140, die ihrerseits in Zeile 220 rutscht.

```
→ 140 INPUT "WIE VIELE MITSPIELER";MS
  150 SP=1
```

MS ist also die festgelegte Mitspielerzahl, die während eines Spiels konstant bleibt. Da wir aber mit Spieler 1 anfangen und mit Spieler M$ aufhören, brauchen wir noch eine Spieler-Zählvariable, die pro Durchlauf hochgezählt wird. Ich nenne sie »SP« und setze sie in Zeile 150 auf 1.

Als nächstes folgt die große Schleife für die Spieldurchgänge pro Mitspieler. Ihre Zahl ist natürlich von MS abhängig:

```
→ 70 SP=SP+1
  280 IF SP<> MS+1 THEN 210
```

Die Schleife zwischen den Zeilen 210 und 280 läuft so lange, bis SP, welches am Anfang 1 ist und nach jedem Durchgang in Zeile 270 um 1 weitergezählt wird, die volle Mitspielerzahl MS erreicht hat (Prüfung in Zeile 210 auf MS+1).

In die Schleife lege ich jetzt die Anweisung der alten Zeile 140 mit der Spielanleitung und welcher Mitspieler an der Reihe ist:

```
→ 210 PRINT"MITSPIELER "SP" IST AN DER REIHE"
  220 PRINT"NENNE " S$ ", DIE MIT " B$
      "ANFANGEN"
```


Nach Zeile 220 muß jetzt der Sprung auf den Spielteil erfolgen, den wir ans Ende des Programms zwischen den Zeilen 1230 und 1340 geschoben haben.

Bislang sind wir auf solche Stellen mit GOTO 1230 hin- und mit GOTO xxx wieder zurückgesprungen. Basic kennt aber einen Befehl, der das viel eleganter macht, den sogenannten »Unterprogrammsprung«. Er heißt

GOSUB-RETURN

wobei »Sub« vom englischen »Subroutine« kommt.

Der Befehl GOSUB steht anstelle des GOTO, RETURN kommt ohne weitere Angaben an den Schluß des anzuspringenden Programmteils, also dorthin, wo wir das zweite GOTO hingesetzt hätten. In unserem Fall sieht das so aus:

```
→ 230 GOSUB 1230
  1350 RETURN
```

Wichtig ist noch zu erwähnen, daß der Programmteil zwischen den Zeilen 1230 und 1350 Unterprogramm heißt.

Wichtig ist außerdem, daß der RETURN-Befehl dann automatisch auf die Zeile nach dem GOSUB springt.

Sinn und Zweck der Unterprogramm-Technik ist, Programmteile zu bilden, die man nicht nur innerhalb eines Programms mehrfach verwenden kann, sondern die auch in sich beliebig verändert werden können, ohne das Hauptprogramm zu stören. Alles, was bekannt beziehungsweise konstant bleiben muß, ist die Anfangs-Zeilennummer des Unterprogramms.

Ein weiterer Vorteil eines Unterprogramms ist, daß es einzeln auf Band oder Diskette in einer »Unterprogramm-Bibliothek« gespeichert und in anderen Programmen eingesetzt werden kann, ohne es immer wieder neu programmieren zu müssen.

Wir erklären den Programmteil »STOP & ERGEBNIS« ab Zeile 1430 ebenfalls zum Unterprogramm und springen

nach dem ersten Unterprogramm »ZEITSCHLEIFE & EINGABE« mit einem zweiten GOSUB-Befehl dorthin. Voraussetzung ist, daß es mit RETURN abgeschlossen ist:

```
→ 240 GOSUB 1430
  1500 RETURN
```

Wenn Sie das Programm jetzt schon laufenlassen, werden Sie sehen, daß nur noch wenig fehlt. Zum einen werden die Ergebnisse der einzelnen Mitspieler aufaddiert, zum zweiten fehlt noch eine Rücksetzung vor jedem neuen Spieler, und als letztes brauchen wir noch ein Gesamtergebnis. Diese Dinge sind aber nichts Neues für uns.

```
→ 250 PRINT CHR$(147)
  260 R=0
```

Diese beiden Zeilen besorgen das Löschen und die Rücksetzung des Einzelresultats R. Dieses R müssen wir aber speichern, um es am Ende ausdrucken zu können. Das machen wir am Schluß des zweiten Unterprogramms, wo wir ja in Zeile 1450 das Einzelergebnis R ausdrucken.

Das Gesamtergebnis besteht also aus soviel Einzelergebnissen, wie Mitspieler vorhanden sind. Das schreit nach einem Feld!

Da sicher weniger als elf Mitspieler vorzusehen sind, brauchen wir das Feld nicht zu dimensionieren. Wir teilen direkt einer Feldvariablen R(SP) die jeweiligen Einzelresultate zu, wobei SP ja von 1 bis MS hochgezählt wird. Dadurch werden alle anfallenden Werte von R gespeichert.

```
→ 1450 R(SP)=R
  1460 PRINT R(SP)
  1470 PRINT"ZUR FORTSETZUNG TASTE DRÜCKEN"
  1480 GET A$:IF A$="" THEN 1480
```

Im Schlußteil wird das Endergebnis ausgedruckt. Es steht, wie gesagt, in einem Feld R(0), R(1),...R(letzter Spieler), das wir mit einer FOR-NEXT-Schleife ausdrucken:

```
→ 430 FOR I=1 TO (SP-1)
  440 PRINT"SPIELER "I" HAT: "R(I)
  450 NEXT I
  460 END
```

Zeile 460 ist wichtig, da sie das Hauptprogramm von den Unterprogrammen trennt.

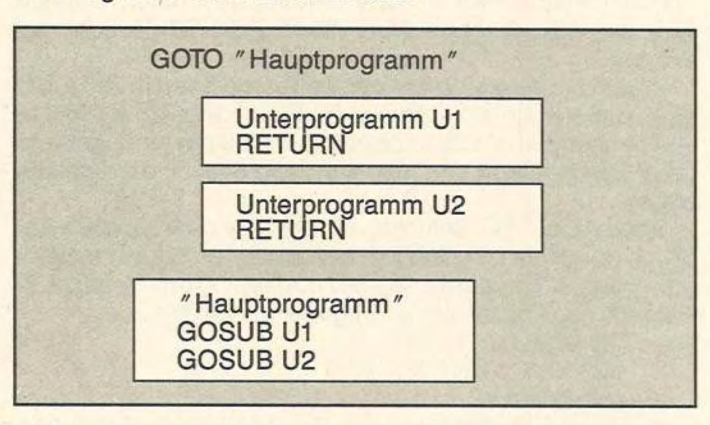
Dieses Programm hat uns also in die Technik der Unterprogramme eingeführt, außerdem haben wir durch eine Schleife mit Abfrage der inneren Uhr des C64 eine Zeitsteuerung des Programms erreicht, haben Farbänderungen eingesetzt und letztlich wieder einmal eine Feldvariable verwendet.

In Listing 9 ist das ganze Programm zusammengefaßt.

Wir haben dort die beiden Unterprogramme an das Ende gelegt. Das war leicht zu erreichen durch einfache Umnumerierung der Zeilen. Wir mußten aber ans Ende des Hauptprogramms – noch vor den Unterprogrammen – einen END-Befehl einfügen, um ein unerwünschtes Weiterlaufen des Programms zu verhindern.

Es gibt nun auch die Möglichkeit, Unterprogramme prinzipiell an den Anfang eines Programms zu legen. Das vermeidet den Weiterlauf und hat außerdem eine kürzere Laufzeit. Ich habe das in dem Kurs »So macht man Programme schneller« im Sonderheft 2/1986 auf Seite 49 gemessen und beschrieben.

Ein Programm sieht dann so aus:



Basic-Befehl Nr. 29 GOSUB-RETURN

- der Befehl wird geschrieben:
  GOSUB Zeilennummer
- er erzeugt einen Sprung in ein Unterprogramm, welches mit der hinter dem GOSUB stehenden Zeilennummer anfängt
- das Unterprogramm muß mit RETURN in der letzten Zeile abgeschlossen werden
- RETURN springt zurück auf die Zeile, die direkt hinter dem GOSUB-Befehl steht
- Unterprogramme können auch geschachtelt werden
- Unterprogramme können sich selbst aufrufen, aber nur bis zu 25mal. Darüber hinaus ist zuwenig Speicherplatz zur Zählung der Aufrufe vorhanden
- trifft ein Programm auf ein RETURN, ohne vorher ein GOSUB gesehen zu haben, reagiert der Computer mit Abbruch und Fehlermeldung »RETURN WITHOUT GOSUB«

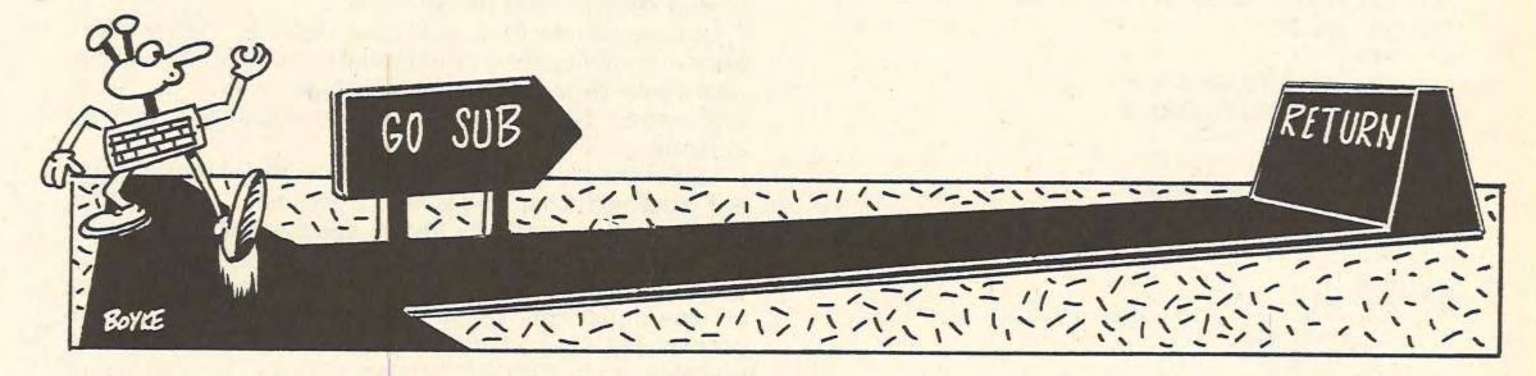
```
100 PRINT CHR$(147)                              <129>
110 INPUT"WAEHLE EIN SACHGEBIET";S$              <235>
120 INPUT"STELLE DIE UHR AUF";Z                  <082>
130 B$=CHR$(INT(RND(0)*26+65))                   <006>
140 INPUT"WIEVIELE MITSPIELER";MS                <197>
150 SP=1                                         <182>
200 REM SCHLEIFE                                 <124>
210 PRINT"SPIELER "SP" IST AN DER REIHE"         <118>
220 PRINT"NENNE "S$" DIE MIT "B$" ANFANGEN
    "                                            <080>
230 GOSUB 1230                                   <058>
240 GOSUB 1430                                   <100>
250 PRINT CHR$(147)                              <023>
260 R=0                                          <231>
270 SP=SP+1                                      <014>
280 IF SP<>MS+1 THEN 200                         <077>
400 :                                            <122>
410 REM ***** ENDERGEBNIS *****                  <080>
420 :                                            <142>
430 FOR I=1 TO (SP-1)                            <074>
440 PRINT"SPIELER "I" HAT: "R(I)                 <245>
450 NEXT I                                       <024>
460 END                                          <208>
1200 :                                           <160>
1210 REM**** ZEITSCHLEIFE & EINGABE ****         <013>
1220 :                                           <180>
1230 PRINT"START MIT IRGENDEINER TASTE"          <142>
1240 GET X$:IF X$=""THEN 1240                    <087>
1250 TI$="000000"                                <213>
1260 REM SCHLEIFE                                <168>
1270 GET E$                                      <240>
1280 W$=W$+E$                                    <136>
1290 IF E$<>CHR$(13) THEN 1340                   <200>
1300 IF LEFT$(W$,1)<>B$ THEN 1330                <125>
1310 PRINT W$;                                   <241>
1320 R=R+1                                       <214>
1330 W$=""                                       <177>
1340 IF VAL(TI$)<>Z THEN 1270                    <049>
1350 RETURN                                      <138>
1400 :                                           <106>
1410 REM**** STOP & ERGEBNIS ******              <075>
1420 :                                           <126>
1430 FOR F=0 TO 14                               <254>
1440 POKE 53280,F:NEXT F                         <104>
1450 R(SP)=R                                     <114>
1460 PRINT"ZUR FORTSETZUNG TASTE DRUECKEN"       <203>
1470 GET A$:IF A$=""THEN 1470                    <253>
1500 RETURN                                      <032>
```


Listing 9. Das fertige Stadt-Land-Fluß

Basic-Befehl Nr. 30 ON GOSUB
- der Befehl wird so geschrieben:
  ON Variable GOSUB mehrere Zeilennummern
  wobei die Zeilennummern durch Kommata getrennt sein müssen
- die Variablenwerte legen fest, auf welche Zeilennummer der GOSUB-Befehl springt, und zwar:
  1.0 bis 1.9 1.Zeilennummer hinter dem GOSUB
  2.0 bis 2.9 2.Zeilennummer ...
  3.0 bis 3.9 3.Zeilennummer ...
  und so weiter
- ist die Variable kleiner als 1, springt der GOSUB-Befehl nicht auf eine der Zeilennummern, sondern auf die nach dem ON-GOSUB folgende Zeile
- ist die Anzahl der Variablenwerte größer als die Anzahl der Zeilennummern, verhält sich der ON-GOSUB-Befehl wie beim Variablenwert 0 oder kleiner als 1
- einen negativen Variablenwert quittiert der Computer mit ILLEGAL QUANTITY.
- hinter dem ON-Befehl kann auch eine Formel oder eine Funktion stehen, nur gilt für ihre Werte dasselbe wie für die Werte einer Variablen.

Diese Anordnung ist sicher nur dann empfehlenswert, wenn es auf Geschwindigkeit ankommt, und wenn viele Unterprogramme verwendet werden.

Programme, die mit vielen Unterprogrammen arbeiten, verwenden oft eine eigene Technik, um »bedienungsfreundlich« zu sein. Mit diesem Begriff bezeichnet die Computerwelt Programme, deren Bedienung so ausgelegt ist, daß man von den Details des Programms oder vom Computer selbst praktisch nichts zu verstehen braucht. Ein wesentlicher Bestandteil sind die sogenannten »Menüs«. Menü heißt eigentlich Speisenfolge, im Englischen bedeutet »menü« aber auch Speisekarte und wird darum hier im übertragenen Sinne für Listen oder Aufzählungen verwendet, aus denen man einzelne Punkte auswählen kann.



Ich schreibe diesen Kurs auf meinem C 64 mit einem bekannten Textprogramm, das sich beim Einschalten mit folgendem Menü meldet:

F1 Text bearbeiten
F3 Neuen Text eingeben
F5 Disk Inhalt
F7 Disk Befehle
F8 Ende

Je nachdem, welche Funktionstaste ich drücke, kann ich einen bereits angefangenen Text weiterbearbeiten oder mir anschauen, welche Programme sich auf der Diskette befinden oder spät in der Nacht mit <F8> Schluß machen!

Jede Überschrift bildet ein eigenes Unterprogramm, auf das durch Drücken der vorgegebenen Funktionstaste gesprungen wird. Mit unseren heutigen Kenntnissen würden wir diese fünf Funktionstasten mit fünf IF-THEN-Befehlen abfragen. Größere Menü bräuchten dementsprechend mehr IF-THEN-Zeilen.

In Basic gibt es einen Befehl, mit dem dies sehr viel leichter programmiert werden kann. Er heißt schlicht und einfach

ON

und wird in Verbindung mit dem GOTO oder GOSUB-Befehl verwendet.

Basic-Befehl Nr. 31 ON GOTO
Für ON GOTO gilt das gleiche wie für ON GOSUB, nur werden die Sprünge nach den Regeln des GOTO-Befehls ausgeführt

Ein ON-GOSUB-Befehl sieht so aus:

```
ON X GOSUB 100,150,200,250
```

Diese Zeile ist gleichbedeutend mit:

```
IF X=1 GOSUB 100
IF X=2 GOSUB 150
IF X=3 GOSUB 200
IF X=4 GOSUB 250
```

Die Variable X darf also nur die Werte 1, 2, 3, 4 etc. annehmen. Entsprechend springt der GOSUB-Befehl auf die erste, zweite, dritte oder vierte Zeilennummer.

Hinter dem ON-Befehl kann auch ein Formelausdruck stehen, nur gilt bezüglich seines Resultats das gleiche wie für die Variable. Es ist klar, daß für Entscheidungen zu Unterprogrammsprüngen nicht immer derartige »saubere« Zahlen zur Verfügung stehen. Damit aber die Umrechnung in die Werte 1, 2, 3 etc. nicht aufwendiger als mehrere IF-GOSUB-Befehle wird, werden oft sehr pfiffige Rechen- und Programmierverfahren angewendet. Sie sind es fast immer wert, gesammelt zu werden, wenn man sie in einem Programmlisting antrifft.

Der Befehl ON darf auch in Verbindung mit dem GOTO-Befehl verwendet werden.

25 Positionierung des Cursors

Ich möchte Ihnen noch schnell die Programmierung des obigen Menü zeigen. Ein Menü beginnt immer mit seiner Darstellung auf dem Bildschirm. Das geht einfach über PRINT-Befehle, die nur wenige Zeilen auf dem Bildschirm verteilen müssen. Da es lästig ist, mit programmierten Cursor-Tasten zu operieren, gibt es noch zwei andere Cursorbefehle, die wir gleich anwenden wollen. Der eine heißt

TAB()

abgeleitet von »Tabulator«, der andere heißt

SPC()



was vom englischen Wort »Space«, das heißt »Abstand« kommt. In der Klammer hinter den Befehlen kann eine Zahl von 0 bis maximal 255 stehen.

Beide Befehle werden zusammen mit dem PRINT-Befehl verwendet. Beim TAB-Befehl rückt der Cursor ausgehend vom linken Rand derjenigen Zeile, auf der er sich gerade befindet, um die in der Klammer stehende Anzahl von Leerstellen weiter. Der SPC-Befehl tut dasselbe, aber vom Punkt aus, auf dem sich der Cursor gerade befindet.

```
10 PRINT CHR$(147)
20 PRINT SPC(90)"WAEHLEN SIE BITTE AUS"
```

CHR$(147) löscht bekanntlich nicht nur den Bildschirm, sondern setzt auch den Cursor in die linke obere Ecke. Von dort aus springt er durch SPC(90) 90 Plätze weiter, das sind 80 (also 2 Zeilen) und 10 Plätze. Sie können das leicht nach RUN auf dem Bildschirm abzählen.

```
30 PRINT TAB(202)"(1)   TEXT BEARBEITEN"
```

Nach Ausführung der Zeile 20 steht der Cursor in der 3. Zeile. 202 Plätze vom linken Rand der 3. Zeile aus gerechnet sind 5 Zeilen und 2 Plätze; also steht der Cursor in der 9. Zeile am 2. Platz, und die Klammer vor der 1 wird am 3. Platz gedruckt.

Ich will Ihnen gestehen, ich habe es auch nicht gerechnet, sondern einfach ausprobiert.

Die nächsten Zeilen verwenden in gleicher Weise den SPC-Befehl.

```
40 PRINT SPC(82)"(2)   NEUEN TEXT EINGEBEN"
50 PRINT SPC(82)"(3)   DISK INHALT"
60 PRINT SPC(82)"(4)   DISK BEFEHLE"
70 PRINT SPC(82)"(5)   ENDE"
```

Die Unterprogramme, die wir über das Menü auswählen wollen, lassen wir ab Zeile 500 beginnen. Ich deute sie natürlich nur an:

```
→ 500 PRINT CHR$(147)
  510 PRINT "TEXT BEARBEITEN"
  520 RETURN
  530 :
  540 PRINT CHR$(147)
  550 PRINT "NEUEN TEXT EINGEBEN"
  560 RETURN
  570 :
  580 PRINT CHR$(147)
  590 PRINT"DISK INHALT"
  600 RETURN
  610 :
  620 PRINT CHR$(147)
  630 PRINT"DISK BEFEHLE"
  640 RETURN
  650 :
  660 PRINT CHR$(147)
  670 PRINT"END"
  680 RETURN
```

Die Unterprogramme beginnen also der Reihe nach bei 500, 540, 580, 620 und 660. Dementsprechend lautet der ON-GOSUB-Befehl:

```
→ 110 ON A GOSUB 500, 540, 580, 620, 660
  120 PRINT"NULL"
  130 END
```

Zeile 120 ist die Ziel-Zeile im Fall, daß für A ein Wert kleiner als 1 eingegeben wird.

Die Eingabe mache ich gewöhnlich mit GET A.

In diesem Fall aber, wo ich Ihnen ermöglichen möchte, auch mit krummen Werten von A zu experimentieren, machen wir es mit INPUT.

```
→ 100 INPUT A
```

So einfach ist das. Bitte experimentieren Sie ein bißchen mit diesem Programm.

Basic-Befehle Nr. 32 und 33 TAB(A) SPC(A)
- hinter den beiden Befehlen steht eine Zahl von 0 bis 255. Sie muß in Klammern stehen
- TAB(I) und SPC(I) werden in Verbindung mit dem PRINT-Befehl verwendet
- mit PRINT TAB(I) beginnt der Ausdruck auf dem Bildschirm (oder auf dem Drucker) an dem durch I definierten Platz
  I=0 definiert den linken Rand der Zeile, auf welcher der Cursor sich gerade befindet, mit I=20 werden 20 Plätze vom linken Rand aus übersprungen
- mit PRINT SPC(I) beginnt der Ausdruck um genauso viele Plätze hinter der augenblicklichen Position des Cursors, wie durch I definiert werden
  I=0 bedeutet die augenblickliche Cursorposition, mit I=20 werden 20 Plätze übersprungen
- beide Befehle springen auf den angegebenen Platz, ohne die übersprungenen Plätze zu löschen

Bei dieser Gelegenheit bietet es sich an, einen weiteren Basic-Befehl zu erwähnen, der wie SPC und TAB mit der Position des Cursors auf dem Bildschirm zu tun hat, allerdings in anderer Weise. Er heißt

POS()

was von POSITION abgeleitet ist. Seinem Namen entsprechend liefert er die aktuelle Position des Cursors auf dem Bildschirm.

Die Sache mit dem Cursor ist nicht ganz so genau zu nehmen. Es gibt ja Möglichkeiten innerhalb eines Programms, wo der Cursor überhaupt nicht auftaucht, zum Beispiel bei einer Stringmanipulation. In diesen Fällen zeigt POS die Position hinter dem gerade bearbeiteten Zeichen an. Denken Sie bitte daran, daß die Positionen in einer Zeile von 0 bis 39 gezählt werden. Ein Beispiel macht das viel klarer:

```
PRINT"01234567" POS(0)
```

Dieser Direktbefehl druckt zuerst den String, bestehend aus den Ziffern 0 bis 7 aus und dann die Position hinter dem letzten Zeichen, und das ist die 8.

Übrigens: hinter POS steht eine Klammer. Die Variable – auch Argument genannt – hat keine Bedeutung, es muß nur irgendeine Variable oder ein beliebiger Wert in der Klammer stehen. Sie wird im Fachjargon »Dummy«-Argument genannt.

Leider hat der POS-Befehl eine Einschränkung. Er wirkt nur innerhalb einer »logischen« Zeile, nicht aber auf dem ganzen Bildschirm. Oder anders ausgedrückt, er wirkt nur waagrecht, nicht aber senkrecht, mit Ausnahme der Logischen Zeile. Was das ist, haben wir in Lektion 5 gelernt – es ist eine durch eine Zeilennummer gekennzeichnete Programmzeile, die maximal aus zwei vollen Bildschirmzeilen bestehen kann. Der POS-Befehl gibt als Resultat daher Zahlen von 0 bis 79 aus.

Den Beweis bringen folgende Programmzeilen:

```
10 PRINT CHR$(147)
20 PRINT TAB(238) "AAAAAA" POS(0)
30 PRINT TAB(238) "B" POS(0)
40 PRINT SPC(40) POS(0)
```

Nach den beiden TAB-Befehlen erscheinten die Positionsnummern 44 und 39. Und nach dem SPC-Befehl für 40 Leerstellen erscheint die Positionsnummer 0, denn diese logische Zeile ist gerade 1 Bildschirmzeile lang, so daß die nächste Positionsnummer 0, das heißt, der Anfang der nächsten logischen Zeile ist.

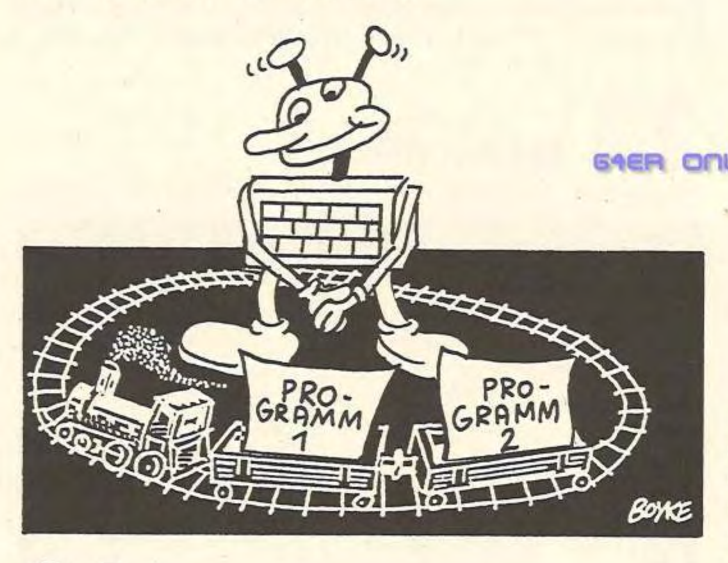
Der POS-Befehl kann natürlich gut für eine Abfrage verwendet werden nach dem Muster:

```
240 IF POS(0) > 25 THEN ......
```

Basic-Befehl Nr. 34 POS (Argument)
- er liefert die aktuelle Position des Cursors auf dem Bildschirm innerhalb einer »logischen« Zeile, das heißt Werte von 0 bis 79
- wird kein Cursor angezeigt, dann bezeichnet er die Position hinter dem zuletzt bearbeiteten Zeichen
- das Argument in der Klammer hinter dem POS-Befehl ist ein »Dummy«-Argument; es muß vorhanden sein, kann jedoch irgendeinen beliebigen Wert haben

Ich habe bei der Aufzählung der Vorteile, welche die Unterprogramm-Technik attraktiv machen, erwähnt, daß es sinnvoll ist, sich eine Bibliothek von nützlichen Unterprogrammen anzulegen, die dann immer wieder in andere Programme eingebaut werden können. Nun, das klingt gut und einleuchtend, aber wie hängt man ein Unterprogramm, das in der »Bibliothek« - sprich auf Band oder Diskette - steht, an ein Hauptprogramm im Computer?

Gute Frage, kann ich nur sagen, denn das Basic der Commodore-Computer kennt leider keinen Befehl dafür. Es gibt ihn, bei anderen Computern oder bei Befehlserweiterungsprogrammen. Er heißt MERGE oder manchmal auch APPEND, was mit »Verschweißen« oder »Anhängen« übersetzt werden kann. Wir müssen ihn in Basic nachvollziehen, was aber nicht schwer ist.



26 Programme zusammenbinden

Wie gesagt, MERGE ist kein Basic-Befehl, sondern ein Kochrezept.

Ich schreibe Ihnen zuerst das Kochrezept auf und erkläre anschließend, was dabei vorgeht. Es werden folgende Schritte gemacht:

1. Ein Programm steht im Speicher (RAM) des Computers. Ein zweites Programm, das auf Band oder Diskette steht, soll dazugeladen werden.

2. Da das Programm auf Band oder Diskette (2. Programm) hinter das Programm im RAM (1. Programm) gehängt wird, ist es empfehlenswert, daß das 2. Programm höhere Zeilennummern als das 1. Programm hat.

3. Die folgende Zeile direkt eingeben und mit <RETURN> abschließen:

```
→ POKE 43,PEEK(45)-2:POKE 44,PEEK(46)
```

4. Jetzt wird das 2. Programm ganz normal mit LOAD in den Computer geladen

5. Die folgende Zeile direkt eingeben und mit <RETURN> abschließen:

```
→ POKE 43,1:POKE 44,8
```

Das ist alles; jetzt steht das 1. Programm und das 2. Programm aneinandergehängt im Speicher des C64.

Zur Erklärung dieses Kochrezeptes muß ich Sie an das Speicherbild in Lektion 21 erinnern, in dem ich die Speicheraufteilung dargestellt habe. Wie dort gezeigt, beginnt der für Programme verfügbare RAM-Speicher ab Adresse 2048 und geht bis 40959.

In diesem Bereich stehen nun die Zeilen eines Programms, alle Namen und augenblicklichen Werte von Variablen, Felder und schließlich auch alle Texte von String-Variablen. Um ein Chaos zu vermeiden und um die Suchzeiten nach Variablenwerten möglichst kurz zu halten, ist dieser Speicherteil in Bereiche unterteilt.

Der Arbeitsspeicher ist unabhängig von seiner durch Speichererweiterung veränderbaren Größe in vier Blöcke und einen Freiraum eingeteilt.
- Block 1 beginnt immer an 2048 und enthält die Zeilen (Text) eines eingegebenen Programms. Sein oberes Ende ist variabel, es hängt von der Länge des Programms ab.
- Block 2 schließt sich direkt an Block 1 an und enthält die numerischen Variablen und die Namen der String-Variablen. Sein Anfang ist identisch mit dem Ende des 1. Blocks. Deshalb ist dieser Anfang ebenfalls variabel. Dadurch ist aber auch das Ende von Block 2 variabel. Wenn zum Beispiel in ein Programm weitere Zeilen eingefügt werden, rutschen alle Grenzen der Speicherbereiche nach oben.
- Block 3 enthält alle Felder.
- Block 4 beginnt am obersten Ende des Arbeitsspeichers und hängt von der Ausbaustufe des Speichers ab.

Er enthält alle Texte der String-Variablen, auch die der Felder. Je nach Länge beziehungsweise Menge der Texte bewegt sich die Grenze des 4. Blocks nach unten, solange bis sie die Grenze des 3. Blocks (die ja nach oben wandert) trifft. Dann ist der Speicher voll, und der Computer meldet OUT OF MEMORY.

Jetzt kommt der Aspekt der Speicheraufteilung, der für unser MERGE-Kochrezept wesentlich ist. Es ist sicher verständlich, daß der Computer, oder besser gesagt seine »Betriebsleitung«, jederzeit wissen muß, wo gerade die verschiedenen Grenzen der Blöcke liegen. Das Betriebssystem besitzt dazu im Systemspeicher (von Adresse 0 bis 1023) einige Speicherzellen, in denen der jeweilige Adressenstand der Blockgrenzen gespeichert ist.

Diese Adressen treten immer paarweise auf. Das liegt daran, daß in einer Speicherzelle allein stets nur Zahlen von 0 bis 255 gespeichert werden können. Das ergäbe 256 Adressen - wir brauchen aber mehr. Wenn man zwei Zellen sozusagen aneinanderhängt, erreicht man dadurch 256*256=65536 Adressen. Diese doppelte Wortlänge - auch Low-/High-Byte-Darstellung genannt - erfordert natürlich einen Rechenvorgang, um aus dem Inhalt von zwei Speicherzellen die dadurch repräsentierte Adresse zu erhalten.

Wie und warum das so ist, erkläre ich ein paar Absätze weiter unten im Abschnitt »Die Low/High-Byte-Darstellung«. Hier ist nur die Umrechnungsformel interessant. Steht in den beiden Speicherzellen X und Y eine Adresse, wird diese ausgerechnet mit:

Adresse = X + 256*Y

Das MERGE-Konzept wollen wir dadurch ausprobieren, daß wir vor das Listing 9 (das Spiel mit Zeitbegrenzung) das Listing 6, nämlich die farbige Umrandung setzten und so ein Programm mit Vorspann daraus machen.

In dem Speicherzellenpaar 43/44 steht also die Anfangsadresse des Arbeitsspeichers. Wir prüfen das nach mit:

```
→ X=PEEK(43):Y=PEEK(44)
  PRINT X:PRINT Y
  PRINT X+256*Y
```

Das Ganze geht natürlich viel kürzer in einer einzigen Zeile:

```
→ PRINT PEEK(43) + 256 * PEEK(44)
```

Wir erhalten beim C64 die Zahl 2049 als Speicheranfang. Entsprechend steht im Speicherzellenpaar 45/46 die Adresse des Endes von Block 1.

Laden Sie bitte Listing 6 in den Speicher und überprüfen das Blockende mit:

```
→ PRINT PEEK(45) + 256 * PEEK(46)
```

Bei mir erscheint da die Zahl 2252, ich hoffe, bei Ihnen auch.

Beim MERGE-Kochrezept führen wir das Betriebssystem des C64 hinters Licht. Mit den Befehlen:

```
→ POKE 43,PEEK(45)-2:POKE 44,PEEK(46)
```

schreiben wir in die Speicherzelle 43 diejenige Zahl, um 2 vermindert, die in Speicherzelle 45 steht. Das gleiche machen wir mit der Speicherzelle 44.

Dadurch verschieben wir mutwillig die Anfangsadresse des Arbeitsspeichers an das Ende des Blocks 1. Da die letzten zwei Speicherzellen hinter einem Programm zur Kennzeichnung des Endes immer nur Nullen enthalten, ziehen wir die 2 ab.

Der Computer glaubt nun, der Arbeitsspeicher beginne bei Adresse 2550 - und ab da lädt er jetzt das Listing 9, ohne Listing 6 zu berühren. Danach allerdings müssen wir die Anfangsadresse wieder an ihren ursprünglichen Platz schieben mit:

```
→ POKE 43,1:POKE 44,8
```


denn 1+256*8 ergibt 2049, was vorher ja dort stand.

Wenn Sie jetzt LISTen, dann stehen beide Programme als ein neues, größeres Programm im Speicher und dieses kann - unter neuem Namen - gespeichert werden.

Speicherzellenpaare wie 43/44, 45/46 nennen wir Zeiger, weil sie auf eine andere Adresse zeigen. Das Hantieren mit Zeigern ist eine sehr reizvolle Sache. Ich bin sicher, Sie werden noch oft damit konfrontiert. Es gibt viele Artikel darüber, ich will mich in diesem Anfängerkurs auf das bisher Gesagte beschränken - es soll ja auch nur eine Einführung sein.

Halt, die doppelte Wortlänge muß ich noch erklären.

27 Die Low-/High-Byte-Darstellung

Ich setze voraus, daß Sie das duale Zahlensystem mit seinen Nullen und Einsen kennen, und daß Sie dadurch wissen, was ein Bit ist, außerdem, wie man Dualzahlen in Dezimalzahlen und umgekehrt umwandelt.

Eine Speicherzelle des C64 hat eine Länge von 8 Bit = 1 Byte.

Mit diesen 8 Bit können Zahlen von 0 bis 255 dargestellt werden. Zur Darstellung von Zahlen über 255 verwenden wir die Low-/High-Byte-Methode.

Wir hängen einfach zwei Speicherzellen zusammen, mit deren 16 Bit wir Zahlen bis maximal 65535 darstellen können. Die maximale Zahl 65535 ist übrigens auch die höchste Adresse des gesamten Speichers - was natürlich kein Zufall ist.

Ich will Ihnen jetzt zeigen, wie eine Dezimalzahl auf zwei 8-Bit-Speicherzellen verteilt wird und umgekehrt: Wie aus 2 Byte eine Dezimalzahl gebildet wird.

Schauen Sie sich das folgende Beispiel an:

DEZIMAL	47491	
DUALZAHL	1011 1001	1000 0011
HIGH BYTE	185	
LOW BYTE		131

Wir gehen von der Dezimalzahl 47491 aus. Ihre duale Darstellung mit 16 Bit - 1011100110000011 - teilen wir einfach in der Mitte und erhalten damit zwei neue Dual-Zahlen mit je 8 Bit = 1 Byte. Das linke Byte nennen wir High-Byte, da es den höheren Teil der Gesamtzahl darstellt. Das rechte Byte heißt entsprechend Low-Byte. Jedes der beiden Bytes kann für sich allein in einer Speicherzelle untergebracht werden, in der natürlich dann der dezimale Wert des Bytes steht. Zur Umrechnung der Low-/High-Bytes empfehle ich folgende Kochrezepte:

Dezimal in Low-/High-Byte

47491:256=185 (High-Byte), Rest 131 (Low-Byte)

Der Rest fällt bei der Division per Hand automatisch an. Mit dem (Taschen)Rechner erhält man den Rest durch:

185*256-47491=-131

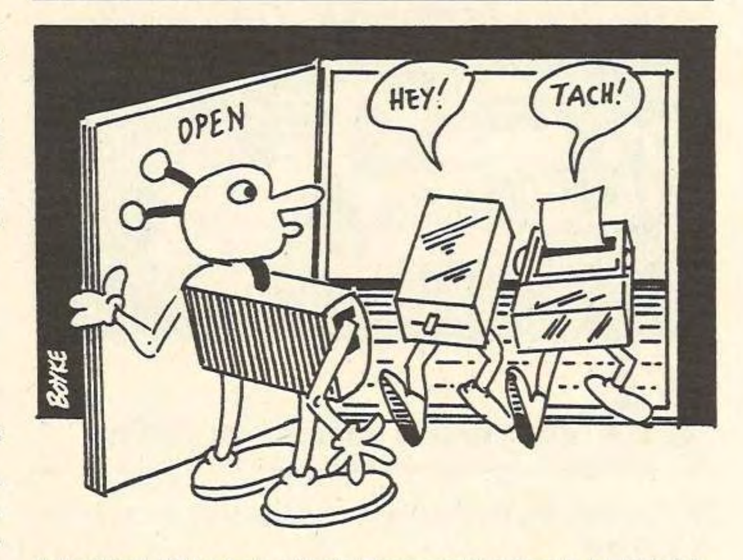
Low-/High-Byte in Dezimal

High-Byte * 256 + Low-Byte = Dezimal

185 * 256 + 131 = 47491

Eine wichtige Regel gilt es noch zu beachten: Der Mikroprozessor des C64 verlangt, daß immer das Low-Byte vor dem High-Byte kommen muß. Die Zahl wird sozusagen von rechts nach links gelesen, in unserem Beispiel 131, 185.

28 Dateien (Files)



In Lektion 20 haben wir ein Programm gebaut, in dem Variablenwerte in ein Feld gespeichert wurden. Wir waren damals noch nicht in der Lage, diese Feldvariablen auf Band oder Diskette abzuspeichern. Ich habe Sie damals gebeten, das Programm-Fragment abzuspeichern, bis wir soweit wären, mehr vom Speicher und vom Speichern zu wissen.

Jetzt sind wir soweit. Mit den Befehlen LOAD speichern wir nur den Inhalt vom 1. Block ab. Die Feldvariablen stecken aber im 3. Block!!

28.1. Der Unterschied zwischen Programm und Datei (File)

Der Computer unterscheidet glücklicherweise beim Speichern auf externe Speichergeräte (Datasette, Diskette)

zwischen Daten aus dem 1. Block und den Blöcken 2, 3 und 4. Daten aus den letzteren nennen wir eine *Datei* oder auf englisch *File*.

Ein Programm besteht aus einer Ansammlung von numerierten Zeilen. Jede Zeile enthält Anweisungen an den Computer. Diese Anweisungen werden nach RUN zeilenweise ausgeführt, in Reihenfolge der aufsteigenden Zeilennummern.

Eine Datei (File) besteht aus rohen Daten, wie Angaben in einem Adreßbuch, ohne jede Angabe, was damit gemacht werden soll. Der bekannte Computerjournalist Richard Mansfield hat zur Erklärung dieses Unterschieds einmal das einleuchtende Beispiel einer Dose mit Tomatensuppe gebracht. Der Aufkleber der Dose enthält sowohl eine Datei als auch ein Programm.

Die Datei lautet:
Wasser, Tomaten, Salz,
Monosodiumglutamat, Farbstoff,
Oleoresin

Das Programm lautet:
1. Vorsichtig öffnen
2. Inhalt in einen Topf leeren
3. eine Tasse Wasser hinzufügen
4. auf kleiner Flamme erhitzen
5. ungewürzt servieren

Ich bin sicher, Sie sehen jetzt den Unterschied.

Programme werden mit SAVE auf Band und auf Diskette gespeichert.

Dateien werden mit einer eigenen Befehlskombination OPEN, PRINT # und CLOSE sowohl auf Band als auch auf Disketten gespeichert.

Genau das wollen wir jetzt lernen.

28.2. Dateien (Files) speichern

Wenn man eine Datei laden oder speichern will, muß zuallererst eine Verbindung zwischen Computer und dem externen Speichergerät hergestellt werden. Ich meine damit nicht das Verbindungskabel, das brauchen wir bei Speichern und Laden von Programmen ja auch. Ich meine hier eine Anweisung an den Computer, sich für den Transfer von Daten über das externe Gerät bereitzuhalten.
Man nennt das auch:

»eine Datei eröffnen«.

Es ist vergleichbar mit dem Herausziehen und Öffnen eines Karteikastens, um Zugriff zu den Daten auf den Karteikarten zu haben.
Der Befehl dazu heißt

OPEN

was natürlich »Öffnen« heißt.

Die zusätzlichen Angaben hinter OPEN bedürfen einer Erklärung. Sie sind sicher für den Anfänger verwirrend, und wir brauchen sie jetzt auch gar nicht alle. Man findet jedoch selten eine komplette Übersicht, daher mache ich die Aufstellung vollzählig – für später.

Basic-Befehl Nr. 35 OPEN
- der OPEN-Befehl öffnet eine Datei zum Schreiben oder Lesen. Mit diesem Befehl wird automatisch ein Pufferspeicher angelegt, mit dem die verschiedenen Arbeitsgeschwindigkeiten der externen Speichergeräte ausgeglichen werden können.
- hinter dem OPEN-Befehl stehen bis zu sechs weitere Angaben, die durch Kommata voneinander getrennt sein müssen. Diese Angaben sind:

OPEN Dateinummer,Gerätenummer,
Sekundäradresse,"Dateiname,Typ,Modus"

Datei-Nummer
ist die Kennzeichnung der eröffneten Datei. Sie kann eine Zahl zwischen 1 und 255 sein. Sie ist wichtig und dient als Referenz beim Umgang mit dieser bestimmtem Datei; dies um so mehr, als gleichzeitig bis zu zehn verschiedene Dateien geöffnet sein können.

Gerätenummer
ist eine Kennzahl dafür, mit welchem externen Gerät die Verbindung hergestellt wird. Dabei gilt:
0 Tastatur
1 Kassettenrecorder (Datasette)
3 Bildschirm
4 Drucker 1
5 Drucker 2, meistens Plotter
8 Diskettenlaufwerk 1
9 Diskettenlaufwerk 2
4 bis 127 Geräte, die am seriellen Bus angeschlossen sind
128 bis 255 Geräte über seriellen Bus, mit einem Zeilenvorschub nach dem Zeilenende.

Diese Angaben sagen Ihnen jetzt noch nichts. Übrigens: Wenn die Gerätenummer weggelassen wird, stellt der Computer automatisch die 1 für Datasette ein.

Sekundär-Adresse
kann eine Zahl zwischen 0 und 255 sein. Diese Zahl hat bei den verschiedenen externen Geräten unterschiedliche Bedeutung.

Tastatur		keine Wirkung
Kassettenrecorder	0	vom Band lesen
	1	auf Band schreiben
	2	auf Band mit »Bandende«-Markierung schreiben
Bildschirm		keine Wirkung
Drucker	0 bis 10	die Funktionen sind bei den Druckerfabrikaten zum Teil unterschiedlich, bitte im Drucker-Handbuch nachsehen
Diskettenlaufwerk		
	0 und 1	vom Betriebssystem reserviert
	2 bis 14	reserviert einen numerierten Datenkanal, bis zu drei gleichzeitig
	15	reserviert einen Befehls-Kanal

Datei-Name
steht immer zwischen Anführungszeichen. Er kann bei der Datasette auch weggelassen werden.

Datei-Typ
wird nur bei Diskettenoperationen verwendet und steht innerhalb der Anführungszeichen hinter dem Datei-Namen; er bezeichnet die folgenden Typen:
S sequentielle Datei
U User-Datei
P Programm-Datei
R relative Datei

Wir werden uns im Rahmen dieses Kurses nur mit sequentiellen Dateien befassen, erstens, weil mit der Datasette nur dieser Typ möglich ist und zweitens, weil er leichter zu verstehen ist.

Modus
wird nur bei Diskettenoperationen verwendet und legt fest, wie der Datenkanal genutzt werden soll:
W Schreiben einer Datei (Write)
R Lesen einer Datei (Read)
A Verlängern einer sequentiellen Datei (Append)
M Lesen einer nicht geschlossenen Datei

Nun bitte keine Panik!

In diesem Kurs brauchen wir nur Bruchteile daraus. Aber immerhin ist das alles ein guter Fingerzeig, daß speziell Diskettenoperationen sehr raffiniert sein können.



Zurück zur Anwendung des OPEN-Befehls. Mit:

```
→ 10 OPEN 1,1,1,"TEXT"
```

eröffnen wir also die Datei "TEXT" auf der Datasette.

Für die Diskette lautet derselbe Befehl:

```
→ 10 OPEN 1,8,3,"TEXT,S,W"
```

S steht für sequentielle Datei, W für Schreiben.

Um jetzt Daten auf das externe Gerät schreiben zu können, verwenden wir eine Variante des PRINT-Befehls, er heißt

PRINT#

Hinter PRINT# muß immer dieselbe Datei-Nummer stehen, die beim OPEN-Befehl verwendet worden ist – sie ist die Referenz. Danach folgt die zu schreibende Variable.

```
→ 20 PRINT#1,"FUSSBALL"
  30 PRINT#1,"TENNIS"
```

Damit haben wir die zwei Wörter abgesendet. Hinter einem PRINT#-Befehl darf immer nur eine Variable oder ein String stehen, damit die Daten in der Datei einen Abstand haben. Als nächstes muß die eröffnete Datei wieder geschlossen werden. Entsprechend zum Eröffnungsbefehl lautet der Schließbefehl CLOSE. Er wird mit der Nummer der Datei versehen.

```
→ 40 CLOSE 1
  50 END
```

Wenn Sie diese vier Zeilen laufenlassen, fordert die Datasette zum Drücken der RECORD- und PLAY-Tasten auf, das Diskettenlaufwerk läuft sofort los und die Datei ist gespeichert.

Dasselbe machen wir jetzt für ein anderes Ausgabegerät, nämlich für den Drucker – mit der Gerätenummer 4.

```
→ 10 OPEN 1,4
  20 PRINT#1,"FUSSBALL"
  30 PRINT#1,"TENNIS"
  40 CLOSE 1
```

Zeile 10 bewirkt, daß die beiden Wörter jetzt vom Drucker ausgedruckt werden.

Basic-Befehl Nr. 36 PRINT#
- der PRINT#-Befehl sendet Daten einer Datei auf Band, Diskette oder Drucker. Maximal können mit einem PRINT#-Befehl 255 Zeichen gesendet werden
- dem PRINT#-Befehl muß dieselbe Dateinummer folgen wie dem OPEN-Befehl, der die Datei eröffnet hat
- es können numerische und String-Variable sowie Feld-Variable gesendet werden. Wichtig ist, daß die einzelnen Variablen durch das RETURN-Zeichen voneinander getrennt sind. RETURN kann entweder durch separate PRINT#-Befehle oder durch CHR$(13) erzeugt werden

Basic-Befehl Nr. 37 CLOSE
- dieser Befehl schließt eine eröffnete Datei
- der CLOSE-Befehl wird lediglich mit der Nummer der Datei versehen, die er schließen soll
- jede Datei muß unbedingt geschlossen werden, da andernfalls keine Endmarkierung hinter die letzte Dateneintragung geschrieben wird. Dadurch können die letzten Daten verlorengehen. In dem Inhaltsverzeichnis (Directory) einer Diskette ist eine nicht geschlossene Datei mit dem Stern »*« gekennzeichnet.

28.3 Dateien (Files) laden

Jetzt kommt der andere Teil – eine auf Band oder Diskette gespeicherte Datei in den Computer laden oder lesen. Die Datei muß wieder eröffnet werden, diesmal zum Lesen.

Für die Datasette lautet der Befehl:

```
100 OPEN 1,1,0,"DATEI"
```

Für die Diskette dagegen muß es heißen:

```
100 OPEN 1,8,3,"DATEI,S,R"
```

Um Daten einer Datei zu lesen, gibt es gleich zwei Befehle, die Sie in ähnlicher Form schon kennen, nämlich

INPUT# und GET#.

Zuerst wenden wir den INPUT#-Befehl an.

In unserer Datei haben wir zwei Strings gespeichert. Mit folgender Zeile lesen wir sie:

```
110 INPUT#1,A$,B$
120 PRINT A$,B$
130 CLOSE 1
```

In Zeile 120 trenne ich das Ausdrucken von A$ und B$ mit einem Komma. Mit dem Klebeeffekt des Semikolon würde nämlich das Wort FUSSBALLTENNIS ausgedruckt werden. Probieren Sie das kleine Leseprogramm mit RUN 100 aus.

Das ganze Listing für Schreiben und Lesen einer String-Datei sieht so aus:

```
10 OPEN 1,1,1,"TEXT"          ...für Band
10 OPEN 1,8,3,"TEXT,S,W"      ...für Diskette
20 PRINT#1,"FUSSBALL"
30 PRINT#1,"TENNIS"
40 CLOSE 1
50 END
60 :
100 OPEN 1,1,0,"DATEI"        ...für Band
100 OPEN 1,8,3,"DATEI,S,R"    ...für Diskette
110 INPUT#1,A$,B$
120 PRINT A$,B$
130 CLOSE 1
```

Für eine Datei mit numerischen Variablen sieht das Programm fast gleich aus:

```
200 OPEN 1,1,1,"ZAHLEN"       ...für Band
200 OPEN 1,8,3"ZAHLEN,S,W"    ...für Diskette
210 PRINT#1,35
220 PRINT#1,14
230 PRINT#1,753
240 PRINT#1,2
250 CLOSE 1
260 END
270 :
300 OPEN 1,1,0,"ZAHLEN"       ...für Band
300 OPEN 1,8,3,"ZAHLEN,S,R"   ...für Diskette
310 INPUT#1,A,B,C,D
320 PRINT A;B;C;D
330 CLOSE 1
```

Bei Zahlen dürfen wir in Zeile 320 ruhig Semikolons verwenden, da Zahlen von selbst Trennungsabstände haben.

Wir wollen noch die Wirkungsweise des GET#-Befehls ausprobieren. Ersetzen Sie im ersten Programm in Zeile 110 das INPUT# durch GET#:

```
110 GET#1,A$,B$
```

Nach RUN 100 erhalten wir den weit auseinandergezogenen Ausdruck der Buchstaben F und U. So geht es also nicht – GET# nimmt immer nur ein Zeichen.

Basic-Befehl Nr. 38 INPUT#
- der INPUT#-Befehl liest, wie GET#, Daten einer mit OPEN eröffneten Datei externer Eingabegeräte
- hinter INPUT# muß die Datei-Nummer des OPEN-Befehls stehen
- während GET# Zeichen für Zeichen liest, holt INPUT# ganze Datengruppen
- hinter einem INPUT#-Befehl können mehrere, durch Kommata getrennte Variable stehen
- ein String darf nicht länger als 80 Zeichen sein
- INPUT# kann nicht im Direkt-Modus verwendet werden

Wir müssen GET# wiederholt einsetzen, am besten mit einer Schleife.

```
110 GET#1,A$
120 PRINT A$;
125 GOTO 110
```

Jetzt erhalten wir die beiden Wörter FUSSBALL und TENNIS, aber die Schleife stoppt nicht. Wir prüfen daher, wann das letzte Zeichen erscheint und CLOSEn dann die Datei. Dazu müssen wir im Eingabeteil (Zeile 30) hinter dem letzten String ein einzigartiges, erkennbares Zeichen einsetzen. Ich habe einfach hinter Tennis ein Leerzeichen gesetzt.

Leider müssen wir dadurch die Datei neu eingeben, am besten mit einem anderen Namen in Zeile 10 und 100. Oder aber Sie löschen die alte Datei »TEXT«.

```
→ 30 PRINT#1,"TENNIS "
  123 IF A$=" " THEN 130
```

Das sind die geänderten Zeilen.

Basic-Befehl Nr. 39 GET#
- der GET#-Befehl funktioniert genauso wie der normale GET-Befehl: Er liest einzelne Zeichen einer geöffneten Datei von einem Eingabegerät. Die Anzahl der Zeichen (Länge der Strings) ist beliebig
- hinter GET# muß die Datei-Nummer des OPEN-Befehls stehen
- GET# kann nicht im Direkt-Modus verwendet werden

Das wiederholte Schreiben der einzelnen PRINT#-Befehle ist natürlich sehr lästig. Besser geht das mit einer FOR-NEXT-Schleife, wobei die zu speichernden Daten entweder direkt per normalen INPUT-Befehl eingegeben oder mit READ aus gespeicherten DATA-Zeilen geholt werden. Ich zeige das an unserer TEXT-Datei in der GET-Version:

```
→ 10 OPEN 1,8,3,"TEXT,S,W"      ...für Diskette
  10 OPEN 1,1,1,"TEXT"          ...für Band
  20 FOR I=1 TO 3
  30 READ A$
  40 PRINT#1,A$
  50 NEXT I
  60 CLOSE 1
  70 DATA FUSSBALL,TENNIS,*
  80 END
  90 :
  100 OPEN 1,8,3,"TEXT,S,R"     ...für Diskette
  100 OPEN 1,1,0,"TEXT"         ...für Band
  110 GET#1,A$
  120 IF A$="*" THEN 130
  123 PRINT A$;
  125 GOTO 110
  130 CLOSE 1
```

Ich habe die Zeile 120, die auf Datei-Ende prüft, leicht geändert. Sie prüft jetzt auf ein abschließendes Zeichen »*«, das ich als letzte Eintragung in die neue DATA-Zeile 70 geschrieben habe. READ und PRINT#1 erfolgen nun innerhalb einer Schleife zwischen Zeile 20 und 50.

Sie müssen leider wieder die Datei TEXT entweder löschen (Diskette) oder überschreiben (Band). Mit RUN läuft der Schreibteil des Programms bis Zeile 80. Mit RUN 100 starten Sie den Leseteil.

Den abspeichernden Teil (also Zeilen 10 bis 70) können wir natürlich auch für die Druckerausgabe verwenden.

Der OPEN-Befehl lautet dafür:

```
210 OPEN 1,4
```

Ab Zeile 220 können Sie die Zeilen 20 bis 70 vom oberen Programm übernehmen, indem Sie eine 2 vor die »alten« Zeilennummern eingeben.

Das Programm sieht jetzt so aus:

```
210 OPEN 1,4
220 FOR I=1 TO 3
230 READ A$
240 PRINT#1,A$
250 NEXT I
260 CLOSE 1
270 DATA FUSSBALL,TENNIS,*
```

Nach RUN druckt der Drucker die Wörter der DATA-Zeile aus. So einfach ist das.

28.4. Kommandogewalt an das externe Gerät

Basic bietet die Möglichkeit an, mit »normalen« Befehlen die angeschlossenen Geräte anzusteuern. Das beste Beispiel dafür ist der Befehl LIST. Wenn er normalerweise das im Programmspeicher des C64 stehende Programm auf dem Bildschirm auslistet, dann soll es möglich sein, mit LIST das Programm auch auf dem Drucker auszudrucken.

Der Befehl dazu ist vom englischen Wort »Command« abgeleitet und heißt

CMD Dateinummer

In Anwendung sieht das, als Direktbefehl eingegeben, so aus:

```
→ OPEN 1,4:CMD 1:LIST
```

Jetzt gilt der LIST-Befehl nicht für den Bildschirm, sondern für den Drucker.

Um die Kommandogewalt wieder dem Bildschirm zu übertragen, muß die Verbindung zum Drucker gelöst werden mit:

```
→ PRINT#1:CLOSE 1
```

Der CMD-Befehl kann auch mit einem Kommentar (wie beim INPUT-Befehl) versehen werden, den er im angesprochenen Gerät »ausdruckt«. Im obigen Beispiel angewendet:

```
→ OPEN 1,4:CMD 1,"LISTING-NAME":LIST
```

Dadurch wird zu Beginn des Listings die Überschrift LISTING-NAME ausgedruckt.

Ein Programm im Speicher kann auch als sequentielle Datei auf die Diskette abgespeichert werden:

```
→ OPEN 1,8,2,"PROGRAMM-NAME,S,W"
→ CMD 1:LIST
```

Jetzt wird das Programm auf der Diskette »ausgeLISTet«, das heißt, es wird sequentiell gespeichert.

```
→ PRINT#1:CLOSE 1
```

Ausgelesen wird es in der schon beschriebenen Weise.

Nicht nur LIST ist ein sinnvoller Befehl, auch PRINT kann anstelle von PRINT# verwendet werden.

```
10 OPEN 1,4
20 CMD 1
30 PRINT"FUSSBALL"
40 PRINT"TENNIS"
50 CLOSE 1
```

Damit werden die beiden Wörter FUSSBALL und TENNIS direkt ausgedruckt – mit PRINT ohne # !

Basic-Befehl Nr. 40 CMD
- er muß immer mit derselben Dateinummer versehen sein, mit der eine Datei durch OPEN eröffnet worden ist.
- die dem CMD-Befehl folgenden Befehle (z.B. LIST oder PRINT) wirken dann auf das angewählte Gerät.
- CMD kann, durch ein Komma getrennt und mit Anführungszeichen versehen, mit einem Kommentar versehen werden, der dann ausgedruckt oder gespeichert wird:
  CMD 1,"KOMMENTAR"
- die Wirkung von CMD kann auf drei Arten aufgehoben werden:
  - CMD 3 (Gerätenummer des Bildschhirms)
  - PRINT#
  - Drücken von <RUN/STOP+RESTORE>



29 Eine Datei speichern

Zum Schluß der Datei-Betrachtungen will ich mein Versprechen einlösen, das ich in Lektion 20 bei den zweidimensionalen Feldern gemacht habe. Ich will Ihnen nämlich zeigen, wie man Werte, die wir in Felder gelegt haben, abspeichern, wieder laden und variieren kann.

Dazu verwenden wir Listing 5, unser Programmbeispiel für zweidimensionale Felder.

Wir wollen schrittweise vorgehen.

→ Laden Sie Listing 5 ein

→ Fügen Sie folgende Zeilen dazu:

```
500 REM---- DATEI ABSPEICHERN -----
505 :
510 OPEN 1,8,4,"LITERATUR,S,W"
520 FOR S=1 TO D
530 FOR K=1 TO 6
540 PRINT#1,A$(S,K)
550 NEXT K:NEXT S
560
670 CLOSE 1
```

Dieser Programmteil speichert, wie Sie nicht nur aus dem Titel, sondern auch aus dem W (write) nach dem OPEN-Befehl sehen können, das gesamte Feld A$ als Datei auf Diskette ab. Für die Datasette muß lediglich Zeile 510 geändert werden:

```
510 OPEN 1,1,1,"LITERATUR"
```

→ RUN

Damit lassen wir das Programm laufen, wodurch die Datensätze der DATA-Zeilen in das Feld eingelesen werden. Der nachfolgende Suchteil des Programms wird jetzt nicht weiter gebraucht und kann durch ein falsches Suchwort und durch »N« bei der Wiederholungsfrage abgebrochen werden.

→ GOTO 500 (nicht RUN 500!)

Genau wie RUN 500 setzt der Direktbefehl GOTO 500 das Programm ab Zeile 500 fort, allerdings ohne die Variablen auf Null zurückzusetzen. Das ist wichtig, damit D seinen letzten, dem aktuellen Stand der Anzahl von Datensätzen entprechenden Wert behält. Mit den Zeilen 500 bis 570 wird das Feld als Datei abgespeichert.

→ Löschen Sie alle DATA-Zeilen

Wir brauchen sie nicht mehr, denn die Datensätze sind ja als Datei gespeichert.

Um die Datei wieder in den Computer laden zu können, müssen wir den READ-Teil des Programm (Zeilen 110 bis 170) umschreiben. Ich schreibe nur die Zeilen, die sich ändern:

```
→ 115 OPEN 1,8,3,"LITERATUR,S,R"
  (115 OPEN 1,1,0"LITERATUR" für Datasette)
  140 INPUT#1,A$(D,K)
  150 IF A$(D,K)=  .....usw..:GOTO 180
  180 CLOSE 1
```

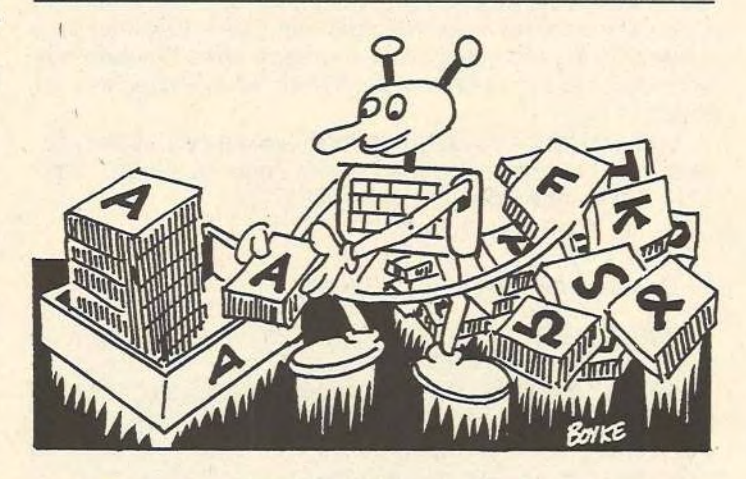
Um das Einlesen der Datei »LITERATUR« ausprobieren zu können, müssen wir das noch im Speicher stehende Feld mit den Datensätzen löschen, am besten mit Aus- und Einschalten des Computers. Vorher speichern Sie auf alle Fälle das unfertige Programm, am besten als Listing 10 A.

→ Computer aus- und wieder einschalten

→ Listing 10 A wieder LOADen und mit RUN laufenlassen

Der Suchvorgang wird Ihnen zeigen und beweisen, daß das Feld tatsächlich als Datei mit dem Namen »LITERATUR« auf der Diskette (Kassette) gespeichert ist.

Merken wir uns:
1. der Direktbefehl RUN 500 setzt alle Variablen auf den Wert 0 zurück und beginnt das Programm ab Zeile 500
2. der Direktbefehl GOTO 500 setzt das Programm ebenfalls ab Zeile 500 fort, ohne aber die Variablen zu verändern



30 Ein vollständiges Datei-Programm

Für ein vollständiges Datei-Programm fehlen dem Listing 10 A noch zwei Dinge:
- ein Menü zur freundlichen Bedienung
- eine Möglichkeit, die Datei zu erweitern

Das Menü soll uns helfen, das ganze Programm, das ja schon fast vollständig ist, zu strukturieren.

Das Menü ist genauso aufgebaut wie das Beispiel in Lektion 24.

```
5 REM---- MENUE -----
7 :
10 PRINT CHR$(147)
15 PRINT SPC(90)"WAEHLEN SIE BITTE AUS"
20 PRINT TAB(202)"(1) DATEN LADEN"
25 PRINT SPC(82)"(2) DATENSAETZE SUCHEN"
30 PRINT SPC(82)"(3) NEUEN TEXT EINGEBEN"
35 PRINT SPC(82)"(4) DATEI ABSPEICHERN"
40 PRINT SPC(82)"(5) ENDE"
45 :
50 PRINT
55 INPUT A
60 ON A GOSUB 100,200,400,500,600
65 GOTO 10
```

Wie gesagt, es ist dasselbe Menü wie vorher, nur der Text ist dem Datei-Programm angepaßt. Auch die Unterprogramm-Zeilennummern in Zeile 60 entsprechen dem Datei-Programm.

Die einzelnen Unterprogramme, soweit sie bereits vorhanden sind, müssen mit RETURN abschließen. Das wollen wir gleich nachholen:

```
→ 190 RETURN
  340 RETURN
  580 RETURN
```

Jetzt fehlen nur noch die Unterprogramme »NEUEN TEXT EINGEBEN« (ab Zeile 400) und »ENDE« (ab Zeile 600)

```
→ 400 REM------ NEUEN TEXT EINGEBEN -------
  410 :
  420 K=1
  430 PRINT K". KATEGORIE":INPUT A$(D,K)
```

```
440 IF K<6 THEN K=K+1:GOTO 430
450 PRINT:PRINT"NOCH EINE EINGABE (J/N) ?"
460 GET V$:IF V$="" THEN 460
470 IF V$="J" THEN D=D+1:GOTO  410
480 GOTO 500
```

Dieses Unterprogramm ersetzt die Eingabe über DATA-Befehle. Die entscheidende Zeile ist Zeile 430. In ihr wird die Eingabe eines kompletten Datensatzes verlangt, wobei die sechs Kategorien numeriert einzeln aufgerufen werden. Diese Eingabe wird sofort einer Feldvariablen A$(D,K) zugewiesen.

Das alles ist nur möglich, weil die Variable D, welche die Anzahl der Datensätze angibt, auch nach dem Suchprogramm wieder auf dem Endwert steht. Sie braucht bei der ersten Eingabe eines neuen Datensatzes nicht verändert werden, weil ihr höchster Wert nicht einen Datensatz, sondern den Klammeraffen »@« gezählt hat. Der aber soll durch den neuen Datensatz überschrieben werden.

Erst wenn in Zeile 470 die Frage nach einer weiteren Eingabe mit J(a) beantwortet wird, muß D um 1 erhöht werden. Wenn nach der Frage kein J(a) erfolgt, springt die Zeile 480 auf das Unterprogramm »DATEI ABSPEICHERN«, damit die Eingabe nicht verlorengeht.

Dieses Unterprogramm speichert die nunmehr vergrößerte Datei natürlich wieder unter dem Namen »LITERATUR« ab. Eine Datei mit diesem Namen existiert aber schon auf der Diskette oder der Kassette. Deshalb muß bei Diskettenbetrieb in Zeile 510 die sogenannte »Save and Replace«-Funktion verwendet werden:

```
→ 510 OPEN 1,8,4,"@:LITERATUR,S,W"
```

Das Einfügen des Klammeraffen »@« gefolgt von einem Doppelpunkt bewirkt, daß die neue Datei die alte Datei desselben Namens überschreibt.

Bei Kassettenbetrieb wird die neue, vergrößerte Datei einfach neu geSAVEt.

Und noch etwas fehlt: die Endmarkierung, die wir in Zeile 150 abfragen und die durch die neue Eintragung überschrieben worden ist.

Sie muß am Ende des Unterprogramms, noch vor dem CLOSE-Befehl, per PRINT#-Befehl an die Datei angehängt werden:

```
→ 560 PRINT#1,"@"
```

Jetzt kommt nur noch das Unterprogramm »ENDE« ab Zeile 600 dazu, das aber denkbar einfach ist:

```
600 REM--------- ENDE ---------
610 END
```

Das komplette Programm ist in Listing 10 wiedergegeben.

```
2 REM+++++++ LITERATUR DATEI ++++++++     <196>
3 :                                          <235>
4 :                                          <236>
5 REM--------- MENUE ----------------        <158>
7 :                                          <239>
10 PRINT CHR$(147)                           <039>
15 PRINT SPC(90)"WAEHLEN SIE BITTE AUS"      <223>
20 PRINT TAB(202)"(1) DATEI LADEN"          <047>
25 PRINT SPC(82) "(2) DATENSAETZE SUCHEN"   <019>
30 PRINT SPC(82) "(3) NEUEN TEXT EINGEBEN"  <253>
35 PRINT SPC(82) "(4) DATEI ABSPEICHERN"    <003>
40 PRINT SPC(82) "(5) ENDE"                 <021>
45 :                                         <021>
50 PRINT                                     <152>
55 INPUT A                                   <087>
60 ON A GOSUB 100,200,400,500,600            <110>
65 GOTO 10                                   <243>
90 :                                         <066>
100 REM-------- DATEI LADEN ----------       <044>
105 :                                        <081>
110 DIM A$(100,6)                            <230>
115 OPEN 1,8,3,"LITERATUR,S,R"               <250>
120 D=D+1                                    <003>
130 FOR K=1 TO 6                             <119>
140 INPUT#1,A$(D,K)                          <064>
150 IF A$(D,K)="@"THEN K=6:NEXT K:GOTO 180   <049>
160 NEXT K                                   <004>
170 GOTO 120                                 <130>
180 CLOSE 1                                  <191>
190 RETURN                                   <248>
195 :                                        <171>
200 REM------ DATENSAETZE SUCHEN ------      <093>
205 :                                        <181>
210 INPUT"STICHWORT";S$                      <148>
220 FOR S=1 TO D                             <024>
230 FOR K=1 TO 6                             <219>
240 IF A$(S,K)=S$ THEN FOR Z=1 TO 6:PRINT
    A$(S,Z):NEXT Z                           <162>
250 NEXT K                                   <094>
260 PRINT                                    <108>
270 NEXT S                                   <180>
280 PRINT"ENDE DER SUCHE"                    <165>
290 PRINT                                    <138>
300 :                                        <022>
310 PRINT"NOCH EINMAL (J/N)?"                <064>
320 GET V$:IF V$="" THEN 320                 <182>
330 IF V$="J" THEN 210                       <054>
340 RETURN                                   <144>
390 :                                        <112>
400 REM----- NEUEN TEXT EINGEBEN ------      <029>
405 :                                        <127>
420 K=1                                      <123>
430 PRINT K".KATEGORIE":INPUT A$(D,K)        <060>
440 IF K<6 THEN K=K+1:GOTO 430               <246>
445 :                                        <167>
450 PRINT:PRINT"NOCH EINE EINGABE (J/N) ?"   <231>
460 GET V$:IF V$="" THEN 460                 <099>
470 IF V$="J" THEN D=D+1:GOTO 420            <080>
480 GOTO 500                                 <186>
490 :                                        <212>
500 REM------- DATEI ABSPEICHERN ------      <220>
505 :                                        <227>
510 OPEN 1,8,4,"@:LITERATUR,S,W"             <217>
520 FOR S=1 TO D                             <072>
530 FOR K=1 TO 6                             <011>
540 PRINT#1,A$(S,K)                          <051>
550 NEXT K:NEXT S                            <085>
560 PRINT#1,"@"                              <070>
570 CLOSE 1                                  <073>
580 RETURN                                   <130>
599 :                                        <067>
600 REM----------- ENDE --------------       <045>
605 :                                        <073>
610 END                                      <104>
```


Listing 10. Die Literatur-Datei

31 Die Booleschen Funktionen AND, OR, NOT

In Lektion 8 bei den Prüfungen habe ich als Prüfbedingungen neben den Vergleichsfunktionen auch die sogenannten Booleschen Funktionen

AND, OR und NOT

erwähnt, ohne aber näher darauf einzugehen. Das will ich hier nachholen. Sie sind prinzipiell bei zwei Verfahren einsetzbar.

31.1. Verwendung als Prüfbedingung

Die Wirkungsweise kann am besten an einem Beispiel demonstriert werden. Ich nehme dazu aus Lektion 8 den vierten Anwendungsfall, bei dem wir zwei IF-THEN-Befehle hintereinander gesetzt haben:

```
350 IF X=4 THEN IF Y=0 THEN PRINT "JA"
```

Diese Zeile kann auch mit AND geschrieben werden:

```
350 IF X=4 AND Y=0 THEN PRINT "JA"
```

Die Wirkungsweise von AND entspricht seiner Übersetzung "UND". Erst wenn beide Bedingungen (X=4 und Y=0) erfüllt sind, wird die Aktion hinter dem THEN-Befehlsteil ausgeführt.

Die OR-Funktion wirkt entgegengesetzt:

```
350 IF X=4 OR Y=0 THEN PRINT "JA"
```

Jetzt wird die Aktion nach THEN dann ausgeführt, wenn entweder die eine Bedingung (X=4) oder die andere (Y=0) oder beide erfüllt sind. Um das auszuprobieren, nehmen wir die gesamte verknüpfte Schleife, aus der die Zeile 350 stammt.

```
310 X=2:Y=1
320 X=X+Y-2
330 Y=Y-X+3
340 PRINT X;Y
350 IF X=4 AND Y=0 THEN PRINT "JA"
360 GOTO 320
```

Das Resultat ist wie gesagt das gleiche wie in Lektion 8. Die Schleife druckt das JA nach dem fünften Umlauf aus.

Wenn Sie in Zeile 350 die OR-Funktion verwenden, wird das JA schon nach dem dritten Umlauf und dann noch einmal nach dem fünften Umlauf ausgedruckt.

Ich will Ihnen noch ein Beispiel zeigen:

```
350 IF (X AND Y)=4 THEN PRINT "JA"
```

Diese Bedingung wird offensichtlich überhaupt nicht erfüllt, denn das JA erscheint nicht. Steht dagegen in der Klammer

```
350 IF (X OR Y)=4 THEN PRINT "JA"
```

dann erscheint das JA, und zwar nach dem fünften Umlauf bei Kombination X=4, Y=0.

Sie sehen also, daß X AND Y nicht gleichbedeutend ist mit X + Y. Das Resultat der beiden Bedingungen ist auf Anhieb nicht verständlich. Das liegt daran, daß diese beiden Funktionen die Variablenwerte im Dualsystem, und da auch noch »Bit-weise« verarbeiten, und zwar nach folgenden Regeln:

UND	ODER
0 AND 0 = 0	0 OR 0 = 0
0 AND 1 = 0	0 OR 1 = 1
1 AND 0 = 0	1 OR 0 = 1
1 AND 1 = 1	1 OR 1 = 1

Ich gehe hier davon aus, daß Sie das duale (oder auch binäre) Zahlensystem, das nur aus den Ziffern 0 und 1 aufgebaut ist, kennen. Ich habe nämlich hier keine Gelegenheit, darauf einzugehen. Wenn Sie es nicht kennen, dann bitte ich Sie, in einem entsprechenden Computerbuch nachzulesen.

Nach obigen Rechenregeln ergibt die Aufgabe

4 AND 3 ergibt 0

4 OR 3 ergibt 7

Im Detail gerechnet sieht das nämlich so aus:

4:0000 0100

3:0000 0011

AND:0000 0000 (entspricht 0)

OR:0000 0111 (entspricht 7)

Jetzt können wir aus den Läufen der Zeilen 310 bis 350 eine Ergebnistabelle machen, die wir auch verstehen können.

	X	Y	X=4 AND Y=0	X=4 OR Y=0	(X AND Y)=4	(X OR Y)=4
Anfang	2	1	nein	nein	nein (0)	nein (3)
1.Lauf	1	3	nein	nein	nein (1)	nein (3)
2.Lauf	2	4	nein	nein	nein (0)	nein (6)
3.Lauf	4	3	nein	JA	nein (0)	nein (7)
4.Lauf	5	1	nein	nein	nein (1)	nein (5)
5.Lauf	4	0	JA	JA	nein (0)	JA (4)
6.Lauf	2	1	nein	nein	nein (0)	nein (3)
und so weiter						

Ganz rechts in der Tabelle stehen die nach den obigen Regeln sich ergebenden Werte in Klammern.

Die dritte Boolesche Funktion

NOT

wird nur sehr selten verwendet. Auch sie verarbeitet die Werte bit-weise im Dualsystem, und zwar vertauscht sie einfach jede 1 mit einer 0 und umgekehrt jede 0 mit einer 1.

5: 0 0000 0101

NOT 5: 1 1111 1010

Diese resultierende Dualzahl 1111 1010 mit noch einer 1 davor entspricht in der speziellen Rechenweise des C64 der negativen Zahl -6.

Auch auf dieses Phänomen - es entsteht durch die spezielle Darstellung der negativen Zahlen im Dualsystem - gehe ich hier nicht näher ein, sondern gebe nur die Formel für NOT an:

NOT X bedeutet -(X + 1)

Sie sehen, eine solche Funktion läßt sich nicht ohne vorbereitendes Kopfrechnen anwenden. Bei der Formel für NOT ist die Klammer sehr wichtig, bei den beiden rechten Bedingungen in der letzten Tabelle auch:

IF (X AND Y) = 4 ist nicht dasselbe wie

IF X AND Y = 4

Probieren Sie es mit der verschachtelten Schleife aus!

31.2. Die Klammer-Regeln

Wir müssen eine weitere Regel beachten, allerdings nicht nur bei den Booleschen Funktionen, sondern auch bei den arithmetischen Funktionen. Anlaß zu dieser Regel bietet die Möglichkeit, in einer IF-Prüfung mehrere Funktionen abzufragen. Es ist, in Erweiterung zu dem was wir ja schon ausprobiert haben, durchaus zulässig, die folgende Bedingung zu machen:

IF A = 1 AND B = 2 OR C = 3 THEN...

Nur muß man beachten, daß diese Zeile so wirkt, als wäre sie mit den folgenden Klammern geschrieben:

IF (A = 1 AND B = 2) OR C = 3 THEN...

Es wird also zuerst die AND-Verknüpfung gebildet und erst dann die OR-Verknüpfung mit dem Resultat der Klammer.

Dieselbe Zeile, wie folgt geschrieben, bringt ein anderes Resultat:

IF A = 1 AND (B = 2 OR C = 3) THEN...

Hier folgt die AND-Verknüpfung erst nach der OR-Verknüpfung. Die Klammern sind deswegen notwendig, weil die einzelnen Funktionen eine eigene »Hackordnung« haben, das heißt, sie haben verschiedene Prioritäten, die in der folgenden Tabelle dargestell sind.

1. ↑ Potenz
2. Vorzeichenwechsel
3. * / Multiplikation, Division (gleichberechtigt)
4. + - Addition, Subtraktion (gleichberechtigt)
5. < > = Vergleichsfunktionen (gleichberechtigt)
6. AND
7. NOT
8. OR

Um Schwierigkeiten zu vermeiden, ist es empfehlenswert, im Zweifelsfall immer Klammern einzusetzen.

31.3. Verwendung als Bit-Maske

In Lektion 21 haben wir beim Stöbern im Speicher gesehen, daß es Speicherzellen gibt, deren Inhalt die Arbeitsweise des Computers beeinflussen. Derartige Speicherzellen nennen wir REGISTER.

So schaltet zum Beispiel die Zahl 5 in das Register 53820 gePOKEt die Farbe des Bildschirmspeichers auf Grün. In den meisten Registern des C64 übt ein einzelnes Bit oder

Bitgruppen eine eigene Steuerfunktion aus. Als Beispiel führe ich das Register 53265 an:

Bit-Nr.	7	6	5	4	3	2	1	0
Stellenwert	128	64	32	16	8	4	2	1

Bit 0–2: bestimmt den Wert der Zeilenverschiebung in Y-Richtung beim Smooth-Scrolling
Bit 3: schaltet Zeilenzahl (0=24 Zeilen, 1=25 Zeilen)
Bit 4: schaltet den Bildschirm (0=gelöscht)
Bit 5: schaltet Hochauflösungsmodus (1=eingeschaltet)
Bit 6: schaltet veränderten Hintergrund-Farbmodus (1=eingeschaltet)
Bit 7: letztes Bit des Farbregisters 53266

Diese einzelnen Funktionen des Registers 53265 werden Ihnen jetzt nicht allzuviel sagen. Wenn Sie mehr über dieses und noch andere Register wissen und lernen wollen, dann verweise ich Sie auf den Aufsatz von Thomas Erhart »Der leichte Umgang mit Sprites« im 64'er Sonderheft 5/86 ab Seite 90. Ich will Ihnen aber zeigen, wie Sie mit Hilfe der Booleschen Funktionen einzelne Bits eines Registers setzen (auf 1) oder löschen (auf 0) können, ohne die anderen Bits zu beeinflussen. An sich ist es ganz einfach. Man nimmt nämlich nur den gegenwärtigen Inhalt des Registers – mit dem PEEK-Befehl – verknüpft ihn über die AND- oder OR-Funktion mit einer MASKE und POKEt das Ganze wieder zurück in das Register.

Was ist eine Maske und wie sieht sie aus?

Bit setzen:

Wenn wir zum Beispiel das 4. Bit des Registers – es hat den Stellenwert 16 – auf 1 setzen wollen, dann hilft uns die OR-Funktion als Maske.

PEEK (53265)	x x x x	x x x x
Maske für 4. Bit	0 0 0 1	0 0 0 0
OR	x x x 1	x x x x

An der Stelle, wo in der Maske eine 1 steht, wird laut Rechenregel (siehe Abschnitt 31.1) durch OR das entsprechende Bit auf 1 gesetzt, alle anderen Bits behalten ihren ursprünglichen Wert.

Der Befehl dazu lautet:

```
POKE 53265,PEEK (53265) OR 16
```

Die Maske zum Setzen des 0. Bit und des 7. Bit schaut demnach so aus:

```
POKE 53265,PEEK (53265) OR (128+1)
```

Bit löschen:

Die Methode ist dieselbe, nur verwenden wir jetzt die AND-Funktion als Maske.

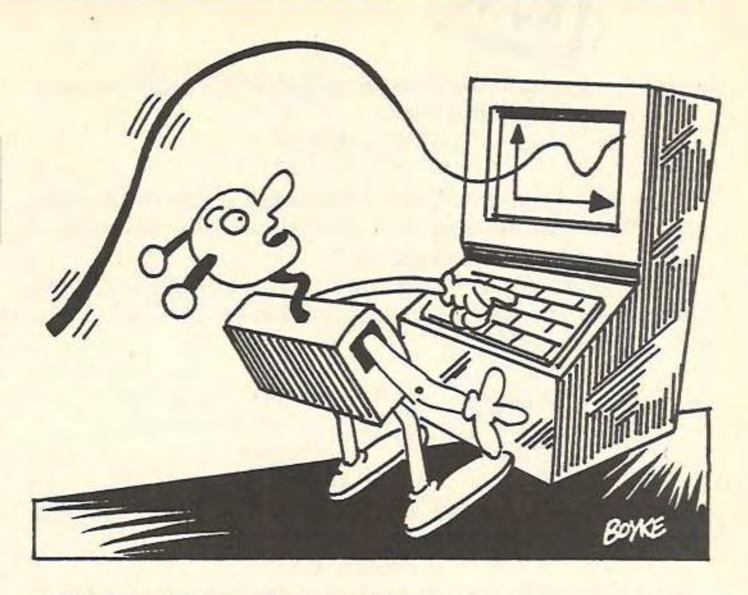
PEEK (53265)	x x x x	x x x x
Maske für 4. Bit	1 1 1 0	1 1 1 1
AND	x x x 0	x x x x

Überall dort, wo in der Maske eine 1 steht, bleibt laut Rechenregel mit AND der ursprüngliche Wert erhalten, nur das Bit mit einer 0 in der Maske wird ebenfalls auf 0 »gelöscht«.

Zur Berechnung der Zahl für die Maske wird einfach der Stellenwert des zu löschenden Bit von 255 abgezogen.

Um Bit 4 zu löschen, lautet der Befehl:

```
POKE 53265,PEEK (53265) AND (255-16)
```



32 Die arithmetischen und geometrischen Funktionen

Diese Funktionen, hinter denen immer in Klammern ein sogenanntes Argument steht, führen mit dem Argument eine bestimmte Berechnung durch.

Alle derartige Funktionen des C 64 finden Sie auch auf einem Schultaschenrechner. Ich werde deshalb – Ihr Wissen voraussetzend – hier nur eine Übersicht geben.

SIN (X)

- berechnet den Sinus von X, wobei das Argument X im Bogenmaß angegeben wird
- wird das Argument in Grad angegeben, so muß folgende Formel verwendet werden:

SIN (X*π/180)

π erzeugen Sie durch die Tastenkombination. <SHIFT> und <↑>

Das folgende Beispiel druckt eine senkrechte Sinuskurve aus:

```
10 PRINT CHR$(147)
20 FOR X=0 TO 6 STEP 0.29
30 A=INT(20+18*SIN(X))
40 PRINT TAB(A);"*"
50 NEXT X
60 END
```

COS (X)

- berechnet den Kosinus von X, wobei das Argument X im Bogenmaß angegeben wird
- wird das Argument in Grad angegeben, so muß folgende Formel verwendet werden:

COS (X*π/180)

Das folgende Beispiel druckt eine senkrechte Kosinuskurve aus:

```
110 PRINT CHR$(147)
120 FOR X=0 TO 6 STEP 0.29
130 A=INT(20+18*COS(X))
140 PRINT TAB(A);"*"
150 NEXT X
160 END
```

TAN (X)

- berechnet den Tangens von X, wobei das Argument X im Bogenmaß angegeben wird
- wird das Argument in Grad angegeben, so muß folgende Formel verwendet werden:

TAN (X*π/180)

ATN (X)

- berechnet den Arcustangens von X, wobei das Argument X im Bogenmaß angegeben wird

- wird das Argument in Grad angegeben, so muß folgende Formel verwendet werden:

ATN (X*π/180)

ABS (X)
- liefert den Absolutwert des Arguments X, das heißt, positive Werte bleiben unverändert, negative Werte werden in positive Werte umgewandelt

Im folgenden Beispiel wird geprüft, ob eine eingegebene Zahl positiv oder negativ ist durch Division der Zahl mit ihrem absoluten Wert.

```
210 INPUT X
220 IF X/ABS(X) =-1 THEN PRINT "NEGATIV"
230 IF X/ABS(X) =1 THEN PRINT "POSITIV"
240 END
```

EXP (X)
- liefert den »natürlichen Exponenten« der Zahl X, oder in Worten ausgedrückt:

e hoch X

(e=2.71828183.. und ist die Basis des natürlichen Logarithmus)
- EXP ist die Umkehrfunktion von LOG.

LOG (X)
- liefert den »natürlichen Logarithmus« (mit Basis e) der Zahl X, die immer positiv sein muß
- LOG ist die Umkehrfunktion von EXP.

SGN (X)
- liefert das Vorzeichen des Arguments X.
- die Funktion kann nur die Werte 1, 0 oder -1 annehmen nach folgender Regel:

bei X = 0 SGN(X)=0
bei X < 0 SGN(X)=-1
bei X > 0 SGN(X)=1

SQR (X)
- berechnet die Quadratwurzel der Zahl X. X muß immer positiv sein
- SQR ist die Umkehrfunktion zum Potenzieren mit ↑2

Im Grunde genommen gehören die Funktionen INT und RND auch in diese Gruppe. Wir haben sie aber bereits in Lektion 10 »Zufalls- und ganze Zahlen« im Detail behandelt.

33 Selbstdefinierte Funktionen

Wem die bisher vorgestellten Funktionen, die uns Basic bietet, nicht genügen, dem erlaubt Basic, seine eigenen Funktionen zu definieren.

Daß ein derartiger Wunsch durchaus vorkommen kann, will ich Ihnen an einigen Beispielen zeigen.

(1) Die Fläche F eines Kreises wird berechnet mit

F=R*R*π

(2) Die Oberfläche O einer Kugel wird berechnet mit

O=R*R*π*4

(3) Das Volumen eines Kegels mit kreisförmiger Grundfläche wird berechnet mit

V=R*R*π*h/4

Der Ausdruck R*R*Pi kommt also recht häufig vor. Wenn Sie jetzt die Aufgabe hätten, für 50 verschiedene Werte des Radius R - R(1) bis R(50) - die entsprechenden Flächen F(1) bis F(50) auszurechnen, dann müßten Sie den Ausdruck R*R*π recht oft verwenden.

Hier lohnt es sich also, dafür eine eigene Funktion zu »erfinden« und ihr einen Namen zu geben, dem man dann wie einer Variablen die verschiedenen Werte zuordnen kann.

Dazu gibt es in Basic den Befehl

DEF FN

(abgeleitet aus DEFine FunktioN).

In unserem obigen Beispiel können wir eine Funktion definieren, der wir den Namen RRPI geben wollen:

DEF FN RRPI(R) = R*R*π

In allen obigen Formeln können wir nun schreiben:

F=FN RRPI(R)
O=FN RRPI(R)*4
V=FN RRPI(R)*h/4

Der eigentliche Pfiff bei dieser selbstdefinierten Funktion liegt in der Eigenschaft der Variablen in der Klammer, in unserem Fall ist es das R. Diese Variable ist nämlich »übertragbar«. Was heißt das?

Wenn wir unsere selbstdefinierte Funktion FN RRPI so verwenden:

X=23+FN RRPI(R)-3

dann lautet die Funktion: R*R*π

Schreiben wir aber:

X=23+FN RRPI(K1)-3

dann lautet die Funktion: K1*K1*π

Die Zuordnung geschieht innerhalb des DEF-Befehls. Leider ist es nicht möglich, mehr als eine derartige übertragbare Variable zu verwenden. Schauen Sie sich das folgende Beispiel an:

```
10 X=20
20 DEF FN Z(R) = 10*INT(2*R+X)
30 X=0.5
40 PRINT FN Z(3.1)
```

In Zeile 20 haben wir in der Formel zwei Variable, R und X. Aber nur R ist als Argument in der Funktion FN Z enthalten. Ihr weisen wir in Zeile 40 den Wert 3.1 zu.

X ist die andere Variable, die vor der Definition der Zeile 20 auf den Wert 20 gesetzt wird. Nach der Definition erhält sie aber den Wert 0.5 (Zeile 30). Und nur das ist der Wert, der in der Berechnung der Zeile 40 verwendet wird:

10*INT(2*3.1+0.5) = 10*INT(6.7)
= 10*6
= 60

Als Einschränkung gilt übrigens, daß man mit DEF FN nur numerische Funktionen definieren darf. Strings haben da keinen Platz.

Neben den wenigen mathematischen Funktionen, die in Lektion 33 aufgeführt sind, gibt es viele andere, die so wichtig sein können, daß Commodore im Handbuch für den C 64 auf Seite 140 eine eigene Tabelle dafür angelegt hat.

Diese Funktionen, die in Basic nicht vorhanden sind, eignen sich natürlich ideal für die Selbstdefinition:

Arcussinus:

DEF FN ARCSIN(X)=ATN(X/SQR(1-X↑2))

Sinus Hyperbolikus:

DEF FN SINH(X)=(EXP(X)-EXP(-X))/2

Interessant ist übrigens, daß es uns durchaus erlaubt ist, eine selbstdefinierte Funktion als Variable (Argument) einer zweiten selbstdefinierten Funktion zu verwenden.

Basic-Befehl Nr. 41 DEF FN
- der Befehl wird so geschrieben:

DEF FN Name(Variable) = Formel
- mit diesem Befehl können eigene komplizierte numerische Funktionen mit einem kurzen Namen definiert werden
- der Name der Funktion beginnt immer mit FN
- der jeweilige Wert der in Klammern stehenden numerischen Variablen wird überall in der Formel eingesetzt, wo die Variable steht
- die Funktion wird nach der Definition aufgerufen mit:

Basic-Befehl FN Name ( )
- DEF FN darf nicht im Direktmodus verwendet werden, String-Variable sind nicht zugelassen

Als Beispiel wähle ich die Formel für das Volumen einer Kugel, deren Ergebnisse ich mit einer anderen Funktion auf zwei Dezimalstellen aufrunde.
Funktion für das Kugelvolumen

```
10 DEF FNV(R)=4*π*R↑3/3
```

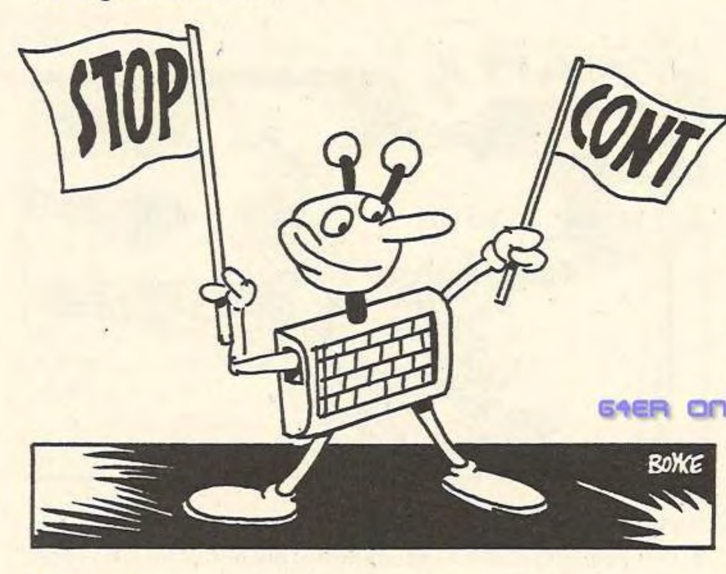
Funktion zum Aufrunden

```
20 DEF FNA(X)=INT((X+0.005)*100)/100
```

Der Rest des Programms lautet:

```
30 INPUT "RADIUS";R
40 PRINT FNV(R);
50 PRINT FNA(FNV(R))
60 GOTO 30
```

Mit diesem kleinen Programm wird in Zeile 40 zuerst das dem eingegebenen Radius entsprechende Volumen der Kugel ausgedruckt und mit dem Semikolon danebengesetzt, danach durch Zeile 50 der aufgerundete Wert. Dabei ist einfach die Volumenfunktion FNV an die Stelle der Variable X gesetzt worden.



34 Fehlersuche

Beim Programmieren gilt ein beinahe ehernes Gesetz: Kein Programm läuft auf Anhieb!
Wie oft haben Sie schon geflucht und gesucht?
Es gibt aber einige Tips und Tricks, wie man sich das Suchen der Fehler erleichtern kann.

Fehlermeldungen

Die häufigsten Fehler treten durch Tippfehler beim Eingeben eines Programms auf. In vielen Fällen reagiert das Übersetzungssystem des Computers mit einer Fehlermeldung, die aber oft nicht verständlich ist. Die Erklärungen der Fehlermeldungen in den Commodore-Handbüchern sind oft recht dürftig. Einige Erläuterungen finden Sie auch auf Seite 129.

Programmunterbrechung

Wenn ein Programm etwas anderes tut, als Sie wollen, ist noch lange nicht klar, wo der Fehler liegt. Um das herauszufinden, ist es empfehlenswert, nur Teile des Programms laufenzulassen. Das geht aber meistens nur vom Anfang an – also halten Sie am besten das Programm an ganz bestimmten Stellen an. Wenn es bis dahin nichts falsch gemacht hat, dann haben Sie den Bereich, in dem der Fehler liegen muß, schon beachtlich eingeschränkt. Zum Anhalten eines Programms gibt es bekanntlich den Befehl END, durch den das Programm abbricht und der Computer sich mit READY meldet.
Es gibt aber noch einen zweiten derartigen Befehl:

STOP

der wie END auch das Programm abbricht, aber mit der Meldung

BREAK IN xxx

die Zeile xxx angibt, wo der STOP-Befehl steht.
Durch diesen Stopp werden die Werte der Variablen nicht verändert und sie können mit einem Direktbefehl ausgedruckt und überprüft werden. Ich komme gleich darauf noch zurück.
Wichtig ist es, daß das gestoppte Programm in der nach dem STOP kommenden Zeile fortgesetzt werden kann, mit GOTO yyy, wenn man die Zeilennummer yyy kennt, oder aber mit dem neuen Basic-Befehl

CONT

was aus dem englischen Wort *Continue* abgeleitet ist. Voraussetzung für das Funktionieren dieses Befehls ist allerdings, daß vorher keine Änderungen im Programm vorgenommen worden sind.
Wenn Sie aber nach dem STOP geLISTet und einen Fehler korrigiert haben, müssen Sie mit RUN wieder von vorn anfangen.
Im Extremfall können Sie in jede Programmzeile ein STOP einbauen und mit CONT zeilenweise das Programm durchgehen, bis Sie zu der kritischen Stelle kommen.
Am Ende, wenn alles läuft, müssen Sie natürlich alle STOPs wieder entfernen.

Variable ausdrucken

Eine Fehleranalyse ist oft dadurch möglich, daß man sieht, welche Werte die verschiedenen Variablen im Lauf eines Programms annehmen.
Einige zusätzliche PRINT-Befehle, zum Beispiel innerhalb einer Schleife, verschandeln zwar den gewünschten Ausdruck auf dem Bildschirm, geben aber Auskunft und Einsicht über den Ablauf und was da so passiert.
Wie oben schon erwähnt, kann man nach einem STOP und ohne ein späteres CONT zu stören, mit PRINT im Direktmodus (also ohne Zeilennummern) den aktuellen Wert einer oder mehrerer Variablen ausdrucken.
Eine etwas seltenere Art der Variablenbehandlung bietet uns ein weiterer Basic-Befehl

CLR

der von CLEAR, also Löschen abgeleitet ist.
Vorsicht! der Befehl CLR hat nichts mit der CLR-Taste gemein.
CLR bezieht sich in seiner »Lösch«-Funktion nur auf den Speicher, während die CLR-Taste den Bildschirm löscht. Mit dem CLR-Befehl werden alle numerischen Variablenwerte auf 0 gesetzt, allen Stringvariablen wird ein »Nullstring« zugewiesen und alle Dimensionen von Feldern werden gelöscht. Wenn DATA-Zeilen vorhanden sind, die ganz oder teilweise gelesen wurden, wird die Funktion von RESTORE ausgeführt. Schließlich werden im RAM-Speicher des Betriebssystems alle vom Programm eingestellten Werte auf ihren Zustand nach dem Einschalten des Computers zurückgesetzt.
Nur zwei Dinge bleiben unberührt – und das ist sehr wichtig: das im Speicher befindliche Programm und Datei-Befehle (OPEN).
Die Funktion von CLR wird übrigens auch bei jedem RUN-, LOAD- und NEW-Befehl ausgeführt.
Der CLR-Befehl kann sehr nützlich sein, um sozusagen die Ausgangslage eines Programms innerhalb eines Programm-Ablaufes wiederherzustellen.



Basic-Befehl Nr. 42 STOP
- dieser Befehl bricht ein Programm ab
- im Unterschied zum END-Befehl meldet sich der Computer nach dem STOP-Befehl mit der Meldung BREAK IN xxx, wobei mit xxx die Zeilennummer angegeben wird, in der das Programm abgebrochen worden ist

Basic-Befehl Nr. 43 CONT
- mit diesem Direktbefehl kann man ein gestopptes Programm, an dem nach dem STOP keine Veränderungen vorgenommen sein dürfen, ab der STOP-Stelle weiterlaufen lassen. Die entsprechende Zeilennummer braucht nicht angegeben zu werden. Alle Werte der Variablen bleiben erhalten

Basic-Befehl Nr. 44 CLR
- dieser Befehl löscht die Werte aller Variablen und Dimensionen von Feldern. Numerische Variable erhalten den Wert 0, String-Variable einen Nullstring. Der Zeiger von READ-DATA-Befehlen wird auf die erste DATA-Zeile gesetzt.
- der Zustand von eröffneten Dateien und ein im Speicher stehendes Programm bleiben unberührt.

35 Seltene Basic-Befehle

In dieser letzten Lektion will ich die Befehle

WAIT, SYS, USR

und die reservierte Status-Variable

ST

behandeln.

Diese vier Kandidaten sind nicht umsonst selten anzutreffen, verlangen doch alle außer WAIT ein fortgeschrittenes Wissen der Programmierung.

Ich bin mir klar darüber, daß für einen derartigen Einführungskurs in Basic SYS, USR und ST zu schwer sind. Ich bringe aber dieses Kapitel der Vollständigkeit halber und entschuldige mich gleich, daß ich es nicht für Einsteiger schreiben kann.

53.1. Der WAIT-Befehl

Normalerweise, wenn in einem Programm gewartet werden soll, verwenden wir eine GET-Schleife vom Typ:

```
10 GET A$:IF A$="" THEN 10
```

In Basic gibt es aber einen – viel zu selten verwendeten – Befehl, der die Programmausführung so lange unterbricht, bis in einer angegebenen Adresse des Speichers etwas verändert wird. Er lautet:

WAIT

Hinter dem WAIT-Befehl, der übersetzt Warten heißt, muß die Adresse einer beliebigen Speicherzelle stehen, in der etwas passieren soll. Zusätzlich folgen auf die Adresse, durch Komma getrennt, eine oder zwei numerische Variable. Was das alles soll, zeigt am besten ein Beispiel.

In Lektion 9 »Eingabe« habe ich den Tastaturpuffer beschrieben, in den Zahlen und Zeichen abgespeichert werden, wenn ihre Tasten während des Programmlaufs gedrückt werden.

In der Speicherzelle 198 nun steht immer die aktuelle Zahl der im Tastaturpuffer gespeicherten Zeichen.

```
100 POKE 198,0
```

Mit dieser Zeile wird »künstlich« der Puffer geleert.

```
110 WAIT 198,1
```

Dieser Befehl unterbricht das Programm so lange, bis irgendeine Taste gedrückt worden ist, wodurch natürlich in Speicherzelle 198 eine 1 steht.

Ein anderes Beispiel bietet die Speicherzelle 653. Sie enthält eine Zahl, die angibt, welche der Steuertasten SHIFT, CTRL oder CBM gedrückt worden ist, und zwar:

SHIFT = 1
CBM = 2
CTRL = 4

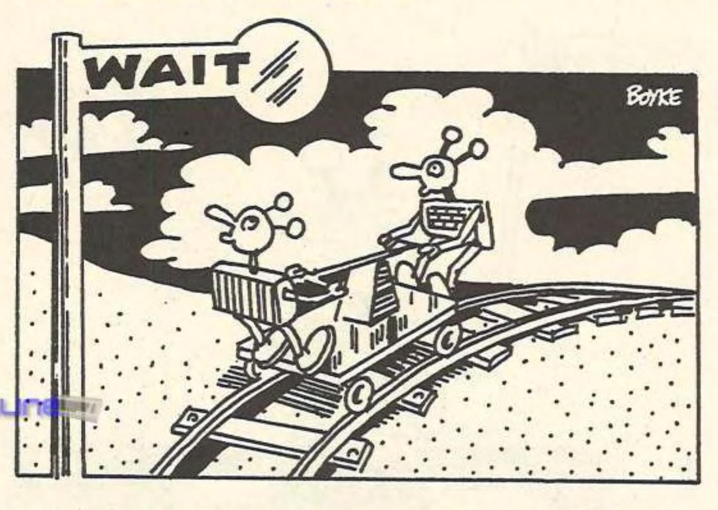
Auch diese Tasten können wir mit WAIT abfragen:

```
200 PRINT "A"
210 WAIT 653,4
230 GOTO 200
```

Mit diesen drei Zeilen wird zuerst ein A ausgedruckt, dann wartet Zeile 210 auf die CTRL-Taste und druckt, solange sie gedrückt ist, lauter A auf den Bildschirm. Um zu erreichen, daß pro Tastendruck nur ein einziges A ausgedruckt wird, fügen wir Zeile 220 ein:

```
220 WAIT 653,4,4
```

Was da passiert, bedarf einer Erklärung, die in der Befehlsbeschreibung (rechts) steht.



Um diese Berechnung zu verstehen, muß ich zuerst die Boole'sche Funktion XOR erklären, die leider im C 64 nicht für den generellen Gebrauch eingebaut ist.

Ähnlich wie das in Lektion 31 beschriebene OR gilt für das XOR folgende Verknüpfungsregel:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Betrachten Sie bitte noch mal die Zeilen 200 bis 230.

```
210 WAIT 653,4
```

Inhalt 653	= 0	0000
N2	= 0	0000
XOR		———
		0000
N1	= 0	0000
AND		———
Ergebnis		0000

das heißt, das Programm wartet.

Wenn die CTRL-Taste gedrückt wird, steht in 653 eine 4. Die obige Rechnung ergibt 4 und das Programm läuft weiter.

Dieselbe Rechnung für Zeile 220 sieht so aus:

```
220 Wait 653,4,4
```

Inhalt 653	= 4	0100
N2	= 4	0100
XOR		———
		0000
N1	= 4	0100
AND		———
Ergebnis		0000

das heißt, das Programm wartet wieder.

Basic-Befehl Nr. 45 WAIT

- Er wird so geschrieben:

  WAIT Adresse,N1,N2
- Er unterbricht ein Programm so lange, bis der Inhalt der mit der Adresse angegebenen Speicherzelle von Null verschieden ist.
- Die Veränderung des Inhalts der adressierten Speicherzelle wird durch die beiden numerischen Variablen N1 und N2 erzeugt, und zwar dadurch, daß zuerst der Inhalt der Speicherzelle mit N2 über ein EXOR verknüpft wird. Dieses Ergebnis wird dann mit N1 über ein AND verknüpft.

  Ist das Resultat ungleich 0, bleibt das Programm unterbrochen.

  Ist das Resultat gleich 0, fährt das Programm mit dem nächsten Befehl fort.
- N2 kann auch weggelassen werden. Dann wird es automatisch auf 0 gesetzt.

35.2. Der SYS-Befehl

Sowohl eigene Programme, die in Maschinensprache geschrieben sind und daher im RAM-Speicher stehen, als auch Programmteile (Routinen) des Betriebssystems und des Basic-Übersetzers, die im ROM-Speicher stehen, können von einem Basic-Programm aus gestartet (aufgerufen) werden. Dieser Befehl entspricht damit dem RUN-Befehl für Basic-Programme. Er wird so geschrieben:

SYS Adresse

Wichtig ist, daß am Ende der mit SYS gestarteten Maschinensprache-Routine ein RTS-Befehl steht, durch den die Programmkontrolle wieder an das Basic-Programm zurückgegeben wird. Ein Beispiel dafür ist

SYS 58260

der Sprung auf den Kaltstart, der den Computer in die Ausgangslage zurücksetzt.

Die meisten dieser Routinen benötigen jedoch verschiedene Angaben - man nennt sie auch Parameter - die vor der Ausführung des SYS-Befehls richtig eingestellt sein müssen.

Die LOAD-Routine zum Beispiel, die ab Speicherzelle 62622, beginnt, können wir mit

SYS 62622



nicht starten. Es fehlen die Angaben über Geräte-Nummer (8 für Floppy, 0 für Band), File-Namen, sowie Anfangs- und Endadresse. Diese Parameter werden normalerweise nach dem Befehl LOAD von der Routine des Interpreters, die den LOAD-Befehl übersetzt, eingegeben. Wir geben ja nicht einfach LOAD ein, wenn wir ein Programm mit dem Namen »TEST« auf Diskette speichern wollen, sondern wir schreiben

```
LOAD "TEST",8
```

Auch wenn wir nur LOAD eintippen, werden vom Übersetzer Parameter gesetzt, nämlich »namenlos« und 0 für Bandgerät. Ich hoffe, Ihnen ist geläufig, daß bei Weglassen aller Angaben der Übersetzer immer Kassettenoperationen durchführt.

Natürlich können wir uns das anschauen: Die Routine des Übersetzers für den BASIC-Befehl LOAD beginnt an Speicherzelle 57704.

Mit SYS 57704 springen wir dorthin - und in der Tat, wir erhalten auf dem Bildschirm PRESS PLAY ON TAPE. Aber ein Programm auf diese Weise auf die Floppy zu LOADen, gelingt uns nicht, es sei denn, wir können die fehlenden Parameter von Hand eingeben.

Genau das aber können wir, weil der SYS-Befehl sich diese Parameter aus den Speicherzellen 780 bis 783 holt und in die vier Register des Mikroprozessors schreibt.

780 ist die Adresse des Akkumulators
781 ist die Adresse des X-Registers
782 ist die Adresse des Y-Registers
783 ist die Adresse des P-Registers.

Die Behandlung von A, X und Y ist unkompliziert, wie wir gleich sehen werden.

Das P-Register ist nicht so einfach zu verwenden, da es nicht Zahlenwerte, sondern Flags (Flagge oder Bitmuster) enthält. Im einzelnen bedeuten im P-Register:

BIT Nr.	WERT	FLAGGE	ABKÜRZUNG
0	1	Übertrag	C(arry)
1	2	Null	Z(ero)
2	4	Unterbrechung	I(nterrupt)
3	8	Dezimal	D
4	16	Abbruch	B(reak)
5	32	nicht benutzt	
6	64	Überlauf	V
7	128	Vorzeichen	N(egativ)

Um eine der Flags des P-Registers zu löschen, empfiehlt es sich, das ganze Register mit POKE 783,0 zu löschen. Umgekehrt muß man beim Setzen der Bits sehr aufpassen wegen der Unterbrechungsflagge I. Eine 1 in I entspricht dem Maschinen-Befehl SEI, der alle Interrupts ausschaltet, auch die der Tastatur-Abfrage, was natürlich sehr störend sein kann! Um alle Flags außer der Unterbrechungsflags I zu setzen, muß POKE 783,247 eingegeben werden.

So, jetzt aber wird es Zeit für ein Beispiel, wie vor dem SYS-Befehl Parameter eingegeben werden können. In der Literatur wird immer das Beispiel gewählt, den Cursor auf eine bestimmte Position zu setzen, beziehungsweise seine Position abzufragen. Dazu gibt es eine Routine, die bei beiden Rechnern ab Speicherzelle 65520 beginnt.

Sie nimmt die Zahl, die im X-Register steht, und verwendet sie als Zeilennummer; die Zahl des Y-Registers nimmt sie als Spaltennummer, setzt dann den Cursor an diese Stelle und bringt die beiden Werte in die Speicherzellen 209/210 und 211.

Unser Beispiel hat die Aufgabe, den Cursor in die 4. Spalte der 7. Zeile zu setzen, dort das Dollar-Zeichen hinzuschreiben und es rot zu färben.

```
5 PRINT CHR$(147)
10 POKE 783,0
20 POKE 781,6
30 POKE 782,3
40 SYS 65520
```

Nach Löschen des Bildschirms werden zuerst alle Flags des P-Registers gelöscht (Zeile 5). Dann kommt die Zeilennummer in das X-Register (Zeile 10) und die Spaltennummer in das Y-Register (Zeile 30). Nach dem Eingeben dieser Parameter können wir mit SYS auf die Routine springen, die ab 65520 beginnt.

```
50 ZEILE=PEEK(209)+256*PEEK(210)
60 ADRESSE = ZEILE + PEEK(211)
70 POKE ADRESSE,36
```

In den Speicherzellen 209/210 können wir jetzt (zur Übung) die Zeilennummer ablesen. Die Adresse der Cursorposition im Bildschirmspeicher erhalten wir durch die Addition der Zeilennummer mit dem Inhalt der Speicherzelle 211. Dorthin POKEn wir den Bildschirmcode des Dollarzeichens, nämlich 36 (Zeile 70).

```
80 SYS 59940
90 FARBE=PEEK(243)+256*PEEK(244)
100 POKE FARBE+PEEK(211),2
```

Für das Färben des Dollarzeichens verwenden wir eine weitere Routine des Betriebssystems, die ab 59940 beginnt. Sie ermittelt die Zeilenposition des Cursors im Farbspeicher und bringt diesen Wert in die Speicherzellen 243/244, wo wir ihn abfragen können (Zeile 90). Die Adresse der Cursorposition im Farbspeicher setzt sich aus diesem Wert plus der Spaltennummer zusammen, die wir wieder der Speicherzelle 211 entnehmen. Auf diesen Platz POKEn wir den Farbcode 2 für Rot (Zeile 100).

So leicht ist das, wenn man die Routinen und die Aufgaben der Speicherzellen kennt.

Basic-Befehl Nr. 46 SYS
- der SYS-Befehl startet ein Maschinenprogramm, das an der hinter dem SYS-Befehl stehenden Adresse beginnt.
- werden mit dem SYS-Befehl Routinen des Betriebssystems aufgerufen, die aber einige Parameter bei der Ausführung brauchen, können diese über die Speicherzellen 780 bis 783 vor dem SYS-Befehl eingegeben werden.

35.3. Der USR-Befehl

Ohne Zweifel gehört auch dieser Befehl zu den Mauerblümchen von Basic, obwohl sein Name – eine Abkürzung von User (Anwender) – eigentlich genügend Anreiz bieten müßte.

USR hat im Grunde genommen dieselbe Funktion wie SYS. Er springt nämlich aus einem Basic-Programm direkt in ein Maschinen-Programm, arbeitet dieses so lange ab, bis er den Befehl RTS findet. RTS entspricht dem Basic-Befehl RETURN und springt in das Basic-Programm zurück.

Bei SYS steht die Sprungadresse gleich hinter dem Befehl.

Bei USR muß die Sprungadresse zuerst in die Speicherzellen 785/786 gePOKEt werden.

Beispiel: Sprung auf 56524
mit SYS: SYS 56524
mit USR: POKE 785,204 (204+256*220=56524)
POKE 786,220
X=USR(Y)

Kein Wunder, daß der Befehl USR sehr selten benutzt wird – ist er doch durch das POKEn der Sprungadresse in Low-/High-Byte-Darstellung aufgebläht.

Das ist aber nicht unnütz, weil USR mehr Fähigkeiten hat als SYS. Im Hinblick auf die im Abschnitt 35.2. »Der SYS-Befehl« aufgezeigten Möglichkeiten des SYS-Befehls sollte ich besser sagen:
USR hat andere Fähigkeiten als SYS.

USR ist eine Mischung von SYS und FN. Letzterer ist der Basic-Befehl zur Definition selbsterfundener Funktionen (Lektion 33).

Bei USR allerdings wird die Funktion als Unterprogramm in Maschinensprache geschrieben, auf die dann wie gesagt der USR-Befehl zur Ausführung springt. Der Pfiff dabei ist aber, daß Zahlenwerte in das Maschinenprogramm mitgenommen beziehungsweise Resultate aus ihm herausgeholt werden können.

Wie läuft das ab: Das Argument Y, das in der Klammer hinter dem Befehl steht, wird zuerst in den Gleitkomma-Akkumulator Nr. 1 (FAC 1) in den Speicherzellen 97 bis 102 gebracht. Als Gleitkommazahl wird es vom angesprungenen Maschinenprogramm weiterverarbeitet. Das Resultat kommt dann wieder in den FAC 1 und steht als Wert von X zur Verfügung.

Das Argument Y kann übrigens auch ein komplexer Ausdruck sein, zum Beispiel:
X=USR(PEEK(A)+256*PEEK(B))

Ich möchte das an einem kleinen Beispiel demonstrieren.

Statt allerdings ein Maschinenprogramm selbst zu schreiben, verwende beziehungsweise springe ich auf eine Routine des Betriebssystems, die den Inhalt des FAC 1 für mathematische Operationen verwendet.

Als geeignete mathematische Operation habe ich die Routine für die Funktion INT gewählt, die im C64 ab der Adresse 48332 beginnt.

Zuerst definieren wir einen Wert für die Variable Y, der in die INT-Routine gebracht werden soll.
10 Y=14.35

Dann bestimmen wir die Sprungadresse für den USR-Befehl. Dazu teilen wir die Adresse 48332 auf in ein Low-Byte = 204 und ein High-Byte = 188. Diese POKEn wir nach 785/786

```
20 POKE 785,204
30 POKE 786,188
```

Jetzt folgt nur noch der USR-Befehl selbst und das Ausdrucken des Resultats.

```
40 X=USR(Y)
50 PRINT X
```

Nach RUN erhalten wir das Resultat 14, wie das Gesetz für INT es befiehlt.

Sie können zur Übung statt INT auch COS verwenden, indem Sie auf die Adresse 57938 springen. Der Vergleich mit dem Befehl COS Y muß dasselbe Ergebnis bringen.

Wer hat übrigens gemerkt, daß wir überhaupt nichts mit der Speicherzelle 784 gemacht haben, obwohl sie doch angeblich am USR-Befehl beteiligt ist? Sie ist es wirklich, doch ohne unser Zutun. In diese Adresse wird beim Einschalten des Computers die Zahl 76 ($4C) geschrieben. Das ist der Code für den Maschinenbefehl »JMP« (JUMP), der dieselbe Wirkung hat wie GOSUB.

Bei Ausführung von USR springt nämlich die entsprechende Routine zuerst auf die Speicherzelle 784, findet dort den Sprungbefehl und in den beiden nachfolgenden Speicherzellen 785 und 786 die Sprungadresse – und führt so den geplanten Sprung aus.

Ich finde, USR ist es wert, in Ihre Überlegungen mit einbezogen zu werden, besonders wenn Sie innerhalb Ihrer BASIC-Programme extrem schnelle Unterprogramme in Maschinensprache eingebaut haben. Diese sind mit USR

Basic-Befehl Nr. 47 USR(X)
- dieser Befehl ruft von einem Basic-Programm aus ein Maschinensprache-Unterprogramm auf
- dabei kann ein numerischer Ausdruck als Argument direkt mitgegeben werden

ganz elegant aufrufbar. Ich denke da zum Beispiel an eine Abfrage der Joysticks.

35.4. Die Status-Variable ST

Neben den Befehlen (wie PRINT) und den Funktionen (wie COS) hat Basic auch noch drei fest definierte Variable, nämlich

TI, TI$ und ST

TI und TI$ haben wir schon in Lektion 23 »Die Uhr des Computers« behandelt.

Von den dreien ist ST wohl am seltensten anzutreffen, Grund genug, hier ein wenig darüber zu berichten.

Die Variable ST gibt den Status nach der letzten Ein- beziehungsweise Ausgabeoperation an, beschränkt allerdings nur auf Operationen mit der Datasette und dem an einem gemeinsamen Ausgang angeschlossenen Diskettenlaufwerk und Drucker.

ST enthält dabei den Inhalt der Speicherzelle 144. Jedes Bit dieses Registers hat eine eigene Bedeutung.

Kassette:
Bit 2 (Wert 4): kurzer Block
Bit 3 (Wert 8): langer Block
Bit 4 (Wert 16): Lesefehler (nicht korrigierbar)
Bit 5 (Wert 32): Prüfsummenfehler
Bit 6 (Wert 64): File-Ende
Bit 7 (Wert 128): Band-Ende

Diskette/Drucker:
Bit 0 (Wert 1): Fehler beim Schreiben
Bit 1 (Wert 2): Fehler beim Lesen
Bit 6 (Wert 64): Daten-Ende
Bit 7 (Wert 128): "Device Not Present"-Fehler
Alle nicht aufgeführten Bits sind nicht benützt.

Wichtig ist noch zu erwähnen, daß nicht nur die in der Tabelle gezeigten Zahlen für ST auftreten, sondern auch Kombinationen davon. So ergibt zum Beispiel ein zu kurzer Block (4) und ein gleichzeitig aufgetretener Prüfsummenfehler (32) einen Wert von 36.

Kassettenoperationen

Zuerst testen wir mit einem Datei-Programm auf »FILE-ENDE«. Geben Sie bitte folgendes Programm ein:

```
10 OPEN 1,1,1,"DATEI"
20 PRINT#1,"QWERTY"
30 CLOSE 1
40 END
```

Zur Erinnerung: Nach dem OPEN-Befehl folgt zuerst die Nummer der Datei (ich nehme hier 1), dann die Gerätenummer (1 = Datasette) und schließlich die Sekundäradresse (1 = Schreiben).

Jetzt kommt der Lesevorgang:

```
50 OPEN 2,1,0,"DATEI"
60 FOR K=1 TO 10
70 GET#2,A$
80 PRINT A$;ST
90 NEXT K
100 CLOSE 2
```

In Zeile 50 eröffnen wir wieder eine Datei (diesmal Nummer 2) für die Datasette, jetzt aber zum Lesen (Sekundäradresse = 0). Die Schleife der Zeilen 60 und 90 schreiben uns 10mal ein Zeichen (A$) und den Wert von ST auf den Bildschirm. Zeile 100 schließt die Datei.

Jetzt geht es los. Mit RUN starten wir den ersten Teil des Programms. Nach dem Schreibvorgang und der READY-Meldung (nach Zeile 40) müssen Sie das Band zurückspulen und mit GOTO 50 ab Zeile 50 weiterfahren. Jetzt wird die Datei gelesen.

Wir erhalten untereinander die sechs Buchstaben von Zeile 20, daneben für ST lauter 0 bis zum Ende, dann allerdings erscheint eine 64. Das ist der in der Tabelle angegebene Wert von ST für »File-Ende«.

Da die FOR-NEXT-Schleife zu lang ist, schießen wir mit dem Lesen über das File-Ende hinaus. Normalerweise kennen wir natürlich die Länge einer Datei nicht. Deshalb ist es besser, mit GOTO zurückzuspringen und das File-Ende abzufragen.

Löschen Sie bitte Zeile 60 und 90 und fügen Sie als Rücksprung und Prüfung ein:

```
85 IF ST=64 THEN 100
90 GOTO 70
```

Statt nach ST können wir natürlich genausogut nach PEEK(144) fragen.

Ein erneutes GOTO 50 bringt das erwünschte Resultat.

Um den vorhin schon erwähnten »kurzen Block« zu sehen, müssen wir einen entsprechenden Fehler künstlich erzeugen.

Löschen Sie bitte den ersten Teil des Programms bis einschließlich Zeile 40. Wir behalten also nur den Leseteil ab Zeile 50. Dann laden wir dieses Programm (Band vorher am besten wieder zurückspulen) mit SAVE »DATEI« nicht als Datei, sondern als ganz gewöhnliches Programm. Wenn es geladen ist, bitte das Band wieder zurückspulen.

Mit RUN starten wir jetzt das Lese-Programm, welches eine Datei sucht, aber nur ein Programm findet, allerdings mit dem richtigen Namen. Natürlich findet es einen Fehler und wir erhalten als Ausdruck:

```
36 oder manchmal auch 4
64 64
```

Die Zahl 36 entsteht aus 32+4, das bedeutet Prüfsummenfehler + Block zu kurz. Danach folgt wie vorher das File-Ende.

Die normale Blocklänge entspricht der Länge des Bandpuffers, in den die Datasette einspeichert. Er ist 191 Byte lang. In unserem Fall war offenbar der Block nicht ganz voll.

Der Prüfsummenfehler tritt dann auf, wenn eine der Überprüfungen von Kassettenoperationen einen Fehler gefunden hat. Der Blockfehler, auch der des zu langen Blocks, interessiert wohl weniger. Aber ein durch die Prüfungen gefundener Fehler könnte, frühzeitig noch vor dem Ausstieg des Programms entdeckt, abgefangen und ausgemerzt werden.

Die Formel dafür, ins obige Programm eingebaut, ist:

```
85 IF ST<64 OR ST>8 THEN ..(z.B. LIST)
```

Statt LIST kann man natürlich auch etwas anderes nehmen.

Diskettenoperationen

Bei dem Diskettenlaufwerk bedeutet ST=64 »DATEN-ENDE«, das ist etwa dasselbe wie bei der Datasette. Um es zu überprüfen, nehmen wir dasselbe Programm wie vorher, nur müssen wir die DATEI-Zeilen der Diskette anpassen. Das sieht dann so aus:

```
10 OPEN 1,8,1,"DATEI,S,W"
20 PRINT#1,"QWERTY"
30 CLOSE 1
40 END
50 OPEN 2,8,0,"DATEI,S,R"
60 FOR K=1 TO 10
70 GET#2,A$
80 PRINT A$;ST
```

```
90 NEXT K
100 CLOSE 2
```

Das Ergebnis sieht hier so aus:

```
64
66
66
```

Die 64 ist natürlich wie erwartet der Wert für Daten-Ende. Die 66 ist 64+2, entstanden dadurch, daß wir über das Daten-Ende hinausgelesen haben. Die 2 bedeutet »Fehler beim Lesen« (in den englischen Beschreibungen heißt es »Read Time Out«). Ähnliches gilt für ST=1, das bedeutet »Fehler beim Schreiben« (englisch: Write Time Out).

Wie bei der Datasette kann das Überlesen natürlich mit der Abfrage

```
IF ST=64 THEN .... und  GOTO...
```

gestoppt werden.

Interessant ist noch der Status beim Fehler »DEVICE NOT PRESENT«, den wir dadurch erzeugen, daß wir ein Programm oder die Directory von der Diskette laden wollen, ohne daß dieses Gerät angeschlossen oder eingeschaltet ist. Nach der Fehlermeldung geben wir direkt ein:

PRINT ST oder PRINT PEEK(144)

und wir erhalten die Zahl 128.

Wie man allerdings in einem Basic-Programm durch Abfrage von ST=128 die Fehlermeldung »Device Not Present« und den dann folgenden Programmabbruch vermeiden kann, bedarf einer gesonderten Maßnahme:

Es gibt zwei Speicherzellen – 768/769 –, in denen in Low-/High-Byte-Darstellung eine Adresse steht, auf die das Betriebssystem springt, wenn die Meldung »DEVICE NOT PRESENT« ausgegeben werden soll. Diesen Zeiger kann man so »verbiegen«, daß die Meldung nicht ausgegeben wird und das Programm einfach weiterläuft.

Normalerweise steht in 768 die Zahl 139, in 769 die Zahl 227. Verbogen wird der Zeiger durch eine 61 (185 geht auch). Dadurch zeigt die Adresse auf eine Speicherzelle des Betriebssystems, in welcher der Assembler-Befehl "RTS", das bedeutet Rücksprung, steht. Jetzt können wir ungestört den ST-atus abfragen, wir müssen allerdings den negativen Wert von ST, also -128 nehmen.

```
10 POKE 768,61            Fehlermeldung abschalten
20 OPEN 1,8,15            Gerät ansprechen
40 CLOSE 1
50 POKE 768,139           Fehlermeldung einschalten
60 IF ST=-128 THEN 100    Sprung bei ausgeschalteten
                          Gerät
70 PRINT" FORTSETZUNG"    weiter im Programm
80 END                    Ende der DEMO
100 PRINT" GERAET EINSCHALTEN"
110 GET A$:IF A$=""THEN 110 Warteschleife
120 GOTO 10               neuer Versuch
```

Drucker-Operation

Für den Drucker sieht das Listing fast genauso aus. Die einzige Änderung ist in den Zeilen 20 und 30:

```
20 OPEN 1,4
30 PRINT#1
```

Ich möchte nach eigenen längeren Versuchen aber dafür plädieren, die Fehlermeldung immer abzuschalten, um nie in Schwierigkeiten zu kommen.
Vorsicht ist die Mutter der Weisheit.

Damit sind wir am Ende unseres Kurses angelangt. Einen Tip will ich Ihnen noch mit auf den weiteren Weg geben: Experimentieren Sie ausgiebig mit den Listings, denn so lernen Sie am schnellsten, welche Möglichkeiten Ihnen die Basic-Programmierung bietet.

Das Ende des Weges?

So, nun haben Sie den Kurs erfolgreich mitgemacht und können sich schon zu den »Basic-Programmierern mit Erfahrung« zählen. Vermutlich fragen Sie sich, wie es weitergehen soll, wie sich Ihre Kenntnisse weiter ausbauen und verfestigen lassen. Darauf gibt es keine Patentantwort. Wir können Sie nur dazu ermutigen, eigene Programme zu entwickeln – denn Übung macht den Meister. Wenn Ihnen dabei ein oder zwei Gleichgesinnte, vielleicht auch jemand mit weiter fortgeschrittenen Kenntnissen zur Seite stehen, um so besser. Reichlich Anregung für Ihre eigenen Projekte finden Sie übrigens regelmäßig in den 64'er-Sonderheften sowie im 64'er-Magazin. Das gilt auch dann, wenn Sie eines Tages in das Programmieren in Maschinensprache einsteigen möchten. Sie sehen, es ist bestens dafür gesorgt, daß Sie Ihr neues Hobby weiter pflegen und entwickeln können. Selbstverständlich besteht auch die Möglichkeit, spezielle Bücher zum Lernen oder Nachschlagen zu Rate zu ziehen. In der Tabelle sind einige Titel aufgeführt, die Ihnen weitere Anregungen und Hilfen vermitteln können.

Wir hoffen, daß Sie noch viel Spaß mit Ihrem Computer haben.

(Dr. H. Hauck/Dr. R. Egg/ef)

Weiterführende Literatur

Wenn Sie Ihren Computer noch besser kennenlernen und Ihre Programmierkenntnisse vertiefen wollen, steht Ihnen eine riesige Auswahl an Literatur zur Verfügung. Damit Ihnen die Auswahl leichtfällt, haben wir einige der interessantesten Bücher zusammengestellt.

H. Tornsdorf, **M.Tornsdorf: C64 Basic für Einsteiger,**
Data Becker, 29 Mark, ISBN 3-89011-246-3

N. Szczepanowski: **64 für Einsteiger**, Data Becker, 29 Mark, ISBN 3-89011-010-X

Willian B. Sanders: **Einführungskurs Commodore 64**, Markt & Technik Verlag, 38 Mark, ISBN 3-89090-607-9

M. Hegenbarth, R.Trierscheid: **Basic Grundkurs mit dem C64**, Markt & Technik Verlag, 44 Mark, ISBN 3-89090-361-4

A. Polk: **Die besten Tips & Tricks**, Data Becker, 29 Mark, ISBN 3-89011-281-1

F. Müller: **C64 Tips, Tricks und Tools**, Commodore Sachbuch, Markt & Technik Verlag, 59 Mark, ISBN 3-89090-499-8

H. J. Liesert: **PEEKS & POKES zum Commodore 64**, Data Becker, 29 Mark, ISBN 3-89011-032-0

Commodore Sachbuch: **Alles über den C64**, Markt & Technik Verlag, 59 Mark, ISBN 3-89090-379-7

Schönleber: **Das Commodore Floppy Buch**, Data Becker, 29 Mark, ISBN 3-89011-269-2

Brückmann, Englisch, Felt, Gelfand, Gerits, Krsnik: **64 Intern**, Data Becker, 69 Mark, ISBN 3-89011-000-2

F. Müller: **C64 für Insider**, Commodore Sachbuch, Markt & Technik Verlag, 59 Mark, ISBN 3-89090-481-5



Illustration: Rolf Boyke

Die 1000 Nöte der Datenspeicherung

Die magnetische Datenspeicherung, sei es auf Kassette oder Diskette, ist eine heikle Sache, bei der hin und wieder Fehler auftreten können. In diesem Bericht klären wir die möglichen Fehler und ihre Ursachen bei Floppy und Datasette.

Die Datasette und die Floppy 1541 sind für Besitzer eines C64 die bekanntesten Medien, um Programme und andere Daten für längere Zeit zu speichern. Sie sind fast schon eine Selbstverständlichkeit geworden. Doch während wir mit diesen Geräten arbeiten, ist uns meist nicht bewußt, welche komplizierten Vorgänge hinter dem Begriff »magnetische Datenspeicherung« stecken. Solange alles ordnungsgemäß abläuft, ahnen wir nichts oder nur wenig von der komplexen Organisation, die uns das Speichern von Programmen auf Kassette oder Diskette ermöglicht. Treten jedoch erste Probleme auf, stellt man fest, daß das eigene Wissen über den C64, die Floppy 1541 oder die Datasette nicht ausreicht.

Das erste Problem, mit dem der Anwender konfrontiert wird, ist die große Anzahl der Fehler, die beim Arbeiten mit dem Computer auftreten können. Doch auch die Floppy 1541 besitzt viele Fehlermeldungen, die verstanden werden sollten. Selbst die Datasette birgt einige Geheimnisse, die für das Laden und Speichern von Daten von Bedeutung sind.

Die Fehlermeldungen der Datasette und der Floppystation sind das Thema dieses Berichts, wobei wir zunächst mit dem einfachen Speichermedium Datasette beginnen wollen.

Soeben haben Sie sich eine Programmkassette gekauft, die ein neues Spiel enthält. Voller Erwartung schalten Sie Ihren C64 ein und laden das Programm. Nach merkwürdig kurzer Zeit stoppt die Datasette unvermittelt, während anders als sonst die ungewöhnliche Meldung »?LOAD ERROR« auf dem Monitor zu lesen ist. Trotz intensiver Bemühungen, wie mehrmaligem Tippen von RUN, läßt sich das Programm nicht starten. Schließlich versuchen Sie, das Programm erneut zu laden, doch erhalten Sie stets die gleiche unangenehme Nachricht auf dem Bildschirm. Das Spiel scheint verloren.

Die eben beschriebene Erfahrung werden viele Computeranwender machen, die oft mit der Datasette arbeiten. Der »?LOAD ERROR« ist dabei eine der vielen Fehlermeldungen, die der Computer zur Verfügung stellt, um Ihnen mitzuteilen, daß bestimmte Vorgänge nicht ordnungsgemäß abgelaufen sind. Er tritt beim Betrieb der Datasette auf und zeigt an, daß das zu ladende Programm nicht einwandfrei von der Kassette gelesen werden kann. Die Ursachen können dabei vielfältiger Art sein, werden jedoch immer mit der schon bekannten Meldung angezeigt.

Unsicheres Speichermedium – Datasette

Für die genauere Lokalisierung eines Fehlers kann eine bereits vom Computer reservierte Variable Auskunft geben. Sie hat den Namen »ST« und wird als Statusvariable bezeichnet, da sie den Status, das heißt, den augenblicklichen Zustand einer Datenübertragung, anzeigt. Jedes der 8 Bit dieses Wertes hat dabei eine bestimmte Aufgabe, welche Sie in Tabelle 1 in einer Übersicht vorfinden. Die Bits 2 bis 5 befassen sich speziell mit der Datasette.

Um Fehler bei der Speicherung so gut als möglich zu vermeiden, trifft der Computer einige Vorkehrungen. Wird ein Programm auf Kassette geschrieben, legt der C64 direkt hinter dem gespeicherten Programm eine komplette Kopie der Daten an, so daß das Programm immer doppelt gespei-

chert wird. Liest der Computer die Daten anschließend wieder ein, lädt er die erste Version in den Speicher. Danach vergleicht er sie mit der Kopie. Stimmen einige Informationen nicht überein, wird zunächst ein Versuch zur Korrektur unternommen. Gelingt dies nicht, setzt der Computer das Bit 4 der Statusvariable (PRINT ST ergibt den Wert 16) und stoppt den Ladevorgang mit einem »?LOAD ERROR«.

Eine weitere Sicherheit bietet eine Prüfsumme. Während des Speicherns bildet der C64 eine Prüfsumme über die Programmdaten und legt sie zusätzlich auf dem Magnetband ab. Laden Sie nun ein Programm, wird aus den gelesenen Bytes ebenfalls eine Prüfsumme errechnet, die mit der gespeicherten übereinstimmen muß. Ansonsten ist wiederum ein »?LOAD ERROR« die Folge. Ein Test der Statusvariable gibt genauere Auskunft. Sie hat in diesem Fall den Wert 32 (Bit 5 gesetzt): ein Prüfsummenfehler.

Zusätzlich kann es vorkommen, daß während des Ladevorgangs besondere Bytemarkierungen nicht ordnungsgemäß erkannt werden. Diese benötigt der Computer zur genauen Synchronisierung beim Lesen von Daten, denn er muß stets den Überblick über den Anfang und das Ende der einzelnen Bytes behalten. Fehlerhafte Markierungen stiften Verwirrung, weil die Daten nicht mehr richtig identifiziert werden können. In diesem Fall hat die Statusvariable ST bei einem »?LOAD ERROR« die Werte 4 oder 8 (Bit 2 oder 3 gesetzt).

Die richtige Einstellung ist alles

Meist hat das Auftreten eines Ladefehlers eine recht einfache Ursache. Im Laufe der Zeit bildet sich eine Schmutzschicht auf dem Tonkopf der Datasette, die das Lesen und Schreiben von Daten behindern kann. Mit einem alkoholgetränkten Wattestäbchen oder einer Reinigungskassette ist eine Reinigung schnell und einfach vorzunehmen. Hat man damit keinen Erfolg, kann das Problem an einem weiteren Phänomen liegen.

Möglicherweise war die Stellung des Tonkopfes Ihrer Datasette bezüglich des Kassettenbandes beim Schreiben des Programmes mit SAVE nicht die gleiche, wie beim Lesen mit LOAD. Man kann dies besonders bei Kassetten beobachten, die nicht mit der eigenen Datasette beschrieben wurden, da die Tonköpfe der Datasetten von Commodore die unterschiedlichsten Einstellungen vorweisen können.

Oft hilft hier ein wiederholter Ladeversuch mit einer etwas anderen Tonkopfjustierung. Das Verstellen läßt sich übrigens leicht an einer der kleinen Schrauben neben dem Tonkopf vornehmen. Eine perfekte Justage ist mit einer kleinen elektronischen Schaltung zu bewerkstelligen, die wir im Sonderheft 15 des 64'er-Magazins vorgestellt haben.

Erscheint nach mehreren Einstellungen jedoch weiterhin die gefürchtete Fehlermeldung, handelt es sich aller Wahrscheinlichkeit nach um eine Kassette mit schadhaftem Bandmaterial. Beim Kauf von Leerkassetten sollten Sie deshalb auf gute Qualität achten (hochwertiges »Lownoise«-Eisenoxidband genügt). Fehlerhafte Softwarekassetten mit fertig gespeicherten Programmen sollten beim entsprechenden Händler reklamiert werden.

Doch auch Ihr bestes selbstgeschriebenes Programm ist möglicherweise noch nicht verloren, wenn ein Fehler aufgetreten ist. Zeigt die Statusvariable die Werte 16 (Fehler bei Vergleich) oder 32 (Prüfsummenfehler) können mit etwas Aufwand und Glück die Daten dennoch gelesen werden, da das Programm aufgrund der Sicherheitskopie zweimal hintereinander auf der Kassette steht. Die Markierungsfehler (PRINT ST ergibt 4 oder 8) sind dagegen recht schwer oder gar nicht zu umgehen. Eine Rettung des Programmes ist hier sehr unwahrscheinlich. Fand der Lesefehler kurz vor Programmende statt, kann bei einem Basic-Programm zumindest der bereits geladene Teil in Sicherheit gebracht werden. Am Ende eines solch verstümmelten Programmes befindet sich ein meist undefinierbares Durcheinander an Werten, was sich bei LIST durch wirre Zeichen in einer Basic-Zeile bemerkbar macht. Ein sogenannter OLD-Befehl (oder auch RENEW: Rückgängigmachen von NEW) kann das »Wirrwarr« wieder in Ordnung bringen. Da der C64 diese Anweisung jedoch bei seinem Basic 2.0 nicht kennt, müssen Sie in diesem Fall auf eine Basic-Erweiterung zurückgreifen.

Damit die bisher geschilderten Probleme nicht auftreten, ist es ratsam, ein gerade gespeichertes Programm sogleich mittels VERIFY auf seine Fehlerfreiheit zu untersuchen. VERIFY vergleicht das im Speicher befindliche Programm mit dem gespeicherten und zeigt eventuelle Unterschiede durch eine Fehlermeldung an. Wurde das Programm korrekt auf Band geschrieben, lautet die Meldung nach dem Vergleich »OK«. Ein »?VERIFY ERROR« bedeutet hingegen, daß Fehler bei der Speicherung oder beim vergleichenden Lesen auftraten. Das Programm sollte nochmals mit SAVE gespeichert werden. Liegt die Vermutung nahe, daß das Kassettenband schadhaft ist, verwenden Sie am besten eine neue Kassette.

Bit	Wert von ST wenn gesetzt	Bedeutung
0	1	Für die Datasette ohne Bedeutung
1	2	Für die Datasette ohne Bedeutung
2	4	zu kurzer Block
3	8	zu langer Block
4	16	Vergleichsfehler von ersten Daten und Kopie
5	32	Prüfsummenfehler
6	64	EOF; Ende der Datenübertragung
7	128	EOT; Bandende

Tabelle 1. Die Bitbelegung der Statusvariable ST

Diese Sicherheitsvorkehrungen sind zwar mit einem großen Zeitaufwand verbunden, vermeiden aber meist Probleme mit den eben beschriebenen Lesefehlern und den Ärger über ein verlorenes Programm. Das Rekonstruieren eines zerstörten Programmes ist zudem sehr viel umständlicher.

Wer jedoch ganz sicher gehen will, sollte zur Speicherung seiner Programme keine Datasette verwenden. Abgesehen von der nicht gerade überwältigenden Arbeitsgeschwindigkeit dieses Gerätes (die sich zwar mit Fast-Tape-Programmen erheblich beschleunigen läßt) ist das Schreiben von Daten auf Kassetten keine besonders sichere Angelegenheit.

Vorbeugen mit VERIFY

Wesentlich sicherer und schneller ist dagegen eine Floppystation, die die Datenspeicherung auf Disketten erlaubt. Abgesehen vom Geschwindigkeitsvorteil bietet sie einen besseren Bedienungskomfort und eine relativ hohe Aufzeichnungsdichte.

Die Floppy 1541 zeigt einen sehr viel komplizierteren Aufbau als die vergleichsweise primitive Datasette. Neben dem Laufwerk und der Mechanik zur Bewegung des Schreib-/Lesekopfes sind viele elektronische Bauteile im Gehäuse des Laufwerks untergebracht. Sie übernehmen die Steuerung der mechanischen Teile und koordinieren den Ablauf des Schreibens und Lesens auf der Diskette. Betrachtet man diese Schaltungen genauer, stellt man fest, daß sie ähnlich kompliziert sind wie die eines Computers, wobei wir auf einen wesentlichen Aspekt der Floppy 1541 zu sprechen kommen.

Im Gegensatz zu anderen Floppylaufwerken, die lediglich Daten lesen und schreiben können, kann die Floppy 1541 als »intelligente« Diskettenstation bezeichnet werden, da sie selbständig Operationen durchführen kann, ohne vom Computer gesteuert werden zu müssen. Wie ein vollständiger Computer besitzt die Floppy 1541 einen eigenen Mikroprozessor (CPU), ein Betriebssystem, das DOS genannt wird, und etwas RAM (Random Access Memory, frei übersetzt Schreib-/Lesespeicher) als Arbeitsspeicher für diverse Aktionen. Lediglich Bildschirm und Tastatur fehlen. Sie werden aber nicht benötigt, da die Kommunikation ausschließlich über Ihren C 64 erfolgt. Das DOS (»Disk Operation System« = Diskettenbetriebssystem) erlaubt dies sogar auf einfache Weise, wie wir noch sehen werden.

Komfort und Sicherheit mit der Floppy 1541

Wie Sie vielleicht wissen, besitzt der C 64 eine Vielzahl (genau 30) verschiedener Fehlermeldungen, die Sie darauf aufmerksam machen, daß zum Beispiel ein Fehler während des Programmablaufs aufgetreten ist. Die »intelligente« Floppy 1541 kennt ebenfalls 34 Fehlermeldungen, die dem Benutzer Ungereimtheiten bei der Arbeit mit einer Diskette mitteilen. Die Floppystation besitzt jedoch keinen Bildschirm, auf dem sie diese anzeigen kann, und so muß sie einen anderen Weg gehen. Ist ein Fehler, etwa beim Lesen eines Programms, aufgetreten, beginnt die sonst konstant leuchtende rote Leuchtdiode am Laufwerk unregelmäßig zu flackern, während ungewöhnliche »Klick«-Geräusche aus dem Inneren des Gehäuses zu vernehmen sind. Oftmals beschwert sich die Floppy 1541 mit einem unangenehmen Rattern, bis der Diskettenmotor schließlich anhält und die rote Leuchtdiode in einem regelmäßigem Rhythmus blinkt. Nach all diesen akustischen und optischen Zeichen weiß der Computeranwender, daß bei der Arbeit mit dem Diskettenlaufwerk ein Fehler erkannt worden ist, doch kann man noch nicht feststellen, um welche Art von Fehler es sich handelt.

Zunächst müssen wir mit der Floppystation in Verbindung treten. Dies geschieht über den sogenannten Kommandokanal der 1541. Er dient in erster Linie zum Übermitteln von Befehlen an die Floppystation, denn als intelligentes Gerät versteht sie natürlich auch Kommandos, die sie prompt und selbständig ausführt. Wir wollen uns aber mit der zweiten Aufgabe des Kommadokanals beschäftigen. Zusätzlich wird er nämlich als »Fehlerkanal« verwendet, der die Status- oder Fehlermeldungen der Floppy 1541 an den Computer übermitteln kann. Ist uns bei der Bedienung der Diskettenstation ein Fehler unterlaufen, was wir an der blinkenden roten LED erkannt haben, können wir über diesen Kanal das Laufwerk nach der Ursache des Mißgeschicks befragen, das heißt die exakte Fehlermeldung erhalten. Doch wie können wir den Fehlerkanal ansprechen?

Die Floppy 1541 besitzt insgesamt 16 Kanäle, die alle gewissen Aufgaben dienen. Sie können entweder zum Schreiben oder Lesen von Daten verwendet werden. Sie sind von 0 bis 15 durchnumeriert, wobei die Kanäle 2 bis 14 dem Anwender zur Verfügung stehen. Die Kanäle 0, 1 und 15 sind jedoch von der Floppystation reserviert, wobei Kanal 15 unser besagter Kommandokanal ist. Er läßt nur das Senden von DOS-Kommandos oder das Empfangen der Fehlermeldungen zu. Wollen wir diesen Kanal ansprechen, müssen wir uns jedoch mit dem nötigen Handwerkszeug vertraut machen. Bevor wir die Fehlermeldung lesen können, müssen wir den Kommandokanal aktivieren, was oft auch als Öffnen bezeichnet wird. Dazu verwenden wir den Basic-Befehl OPEN, den Sie vielleicht schon vom Arbeiten mit Dateien kennen. Für das Öffnen des Kommandokanals ist folgender Befehl einzugeben:

```
OPEN 1,8,15
```

Damit richten wir eine Datei mit der Nummer 1 ein und adressieren die Floppystation mit der Gerätenummer 8. Die letzte Zahl der obigen Anweisung gibt schließlich die Kanalnummer an, die der Datei zugeordnet werden soll. Sie lautet 15, und das ist – wie wir schon wissen – der Kommandokanal.

Nun können wir zum Beispiel einen Befehl an die Floppy 1541 senden, der nach einer Prüfung auf korrekte Syntax ausgeführt wird. Das soll uns aber hier nicht weiter interessieren. Wir wollen nur Daten empfangen, nämlich die Fehlermeldungen, die uns das Laufwerk bereitstellt. Das ermöglichen uns die Anweisungen INPUT# oder GET#. Sie funktionieren ähnlich wie die Anweisungen INPUT und GET. Mit INPUT# können mehrere Zeichen einer Datei gelesen werden, während GET# nur ein Zeichen empfängt. Wie INPUT und GET dürfen beide Befehle nur in einem Programm und nicht im Direktmodus verwendet werden, weshalb wir jeweils eine Zeilennummer voranstellen:

```
10 OPEN 1,8,15
20 INPUT#1,FN,FM$,TR,SE: PRINT FN;FM$;TR;SE
```

Nach Beendigung der Arbeit muß der Kommandokanal wieder geschlossen, das heißt inaktiviert werden, was durch den Befehl CLOSE erfolgt. Schließen wir also unsere Datei mit der Nummer 1 wieder:

```
30 CLOSE 1
```

Wenn Sie die Zeile 20 unseres kleinen Programms betrachten, werden Sie erkennen, daß die Fehlermeldung der 1541 aus vier unterschiedlichen Teilen besteht. Der erste Wert, der über den Kommandokanal empfangen wird, ist die Nummer des aufgetretenen Fehlers (FN). Der darauffolgende String enthält die Meldung im Klartext (FM$), während die abschließenden Werte TR und SE eine nähere Lokalisierung des Fehlers erlauben. Ihre genaue Bedeutung wird im Laufe dieses Berichtes noch erläutert.

Kommunikation mit der Floppy 1541

Ebenso kann die Meldung Zeichen für Zeichen mit GET# gelesen werden. Zeile 20 würde dann so aussehen:

```
20 GET#1,A$: PRINT A$;: IF ST <>64 THEN 20
```

Seltsamerweise tritt hier die uns schon bekannte Statusvariable ST auf. Sie hilft uns, das Ende der Meldung festzustellen. Liegen keine weiteren Daten von einem Gerät am seriellen Bus mehr an, nimmt ST den Wert 64 (Bit 6 ist gesetzt) an. Die vollständige Fehlermeldung wurde gesendet.

Nachdem Sie unser kleines Fehlerprogramm mit RUN gestartet haben, sollten folgende Zeichen ausgegeben werden:

```
0 OK 0 0
```

oder

```
00, OK,00,00
```

Erstere Meldung erhalten Sie bei der Programmversion mit INPUT#, während zweitere bei der Zeile 20 mit GET# ausgegeben wird; die Aussage ist jedoch die gleiche. Diese beiden Texte sind eigentlich keine Fehleranzeigen. Sie teilen uns lediglich mit, daß ein Befehl oder eine Aktion der Floppystation einwandfrei ausgeführt worden ist. Selbst wenn wir kein Kommando gesendet haben, wird nach Ablauf unseres Programms immer diese Statusnachricht auf dem Bildschirm erscheinen. Man kann sie als das »READY« der Floppy 1541 verstehen. Die rote Leuchtdiode blinkt dabei nicht. Lediglich nach dem Einschalten der Diskettenstation können wir eine andere Statusmeldung empfangen. Sie lautet

```
73, CBM DOS V2.6 1541,00,00
```

Damit »begrüßt« uns die Floppy 1541 und sagt uns, mit welcher Version des DOS gearbeitet wird. Daher kann bei einer anderen Floppy eine unterschiedliche Versionsnummer ausgegeben werden. Tritt sie in Verbindung mit einer blinkenden roten Leuchtdiode auf, hat sie eine andere Bedeutung, die wir noch kennenlernen werden. Doch können wir noch eine weitere Statusmeldung empfangen, deren Ankunft nicht durch ein Flackern der Arbeits-LED angekündigt wird.

Das »Wegkratzen« von Dateien

Bei der Arbeit mit Disketten kommt es oftmals vor, daß man ein gespeichertes Programm nicht mehr benötigt. Nun könnte man das Programm durchaus auf der Diskette belassen, jedoch würde dadurch unnötiger Speicherplatz belegt, der für andere Programme oder Daten verlorenginge. Das Programm muß also von der Diskette gelöscht werden. Als eifriger Anwender der Floppy 1541 haben Sie deshalb sicherlich das Handbuch nach einem geeigneten Befehl durchsucht und sind auf eine entsprechende Anweisungsfolge gestoßen:

```
OPEN 1,8,15
PRINT#1,"S:(Name)"
CLOSE 1
```

Haben Sie bisher diese etwas komplizierte Folge einfach als Befehl hingenommen, ohne über die seltsamen Anweisungen nachzudenken, werden Sie spätestens bei der Erklärung des Kommandokanals in diesem Bericht eine Ähnlichkeit zu unserem Fehlerprogramm festgestellt haben. In beiden Fällen muß der Kommandokanal geöffnet werden. Beim Löschen eines Programmes von Diskette wird lediglich mit PRINT# eine Zeichenfolge an die Floppy 1541 geschickt und zwar ein DOS-Befehl mit dem Namen »S«. Er ist die Abkürzung für »SCRATCH«, was auf Deutsch etwa »wegkratzen« bedeutet. Nachdem man den Scratch-Befehl eingegeben hat, beginnt das Laufwerk zu arbeiten (rote Leuchtdiode leuchtet). Nach kurzer Zeit erlischt die rote LED wieder; die Floppy 1541 hat die Datei mit dem angegebenen Namen gelöscht. Als Bestätigung kann man eine Statusmeldung empfangen, die uns mitteilt, daß eine Datei gelöscht wurde:

```
01,FILES SCRATCHED,01,00
```

Die erste Zahl hinter der Klartextmeldung gibt an, wie viele Dateien gelöscht wurden. Dies ist durchaus sinnvoll, da man mit Hilfe der Jokerzeichen (? und *) mehrere Dateien zugleich von der Diskette entfernen lassen kann. Die Anweisung

```
OPEN 1,8,15
PRINT#1,"S:TE*"
CLOSE 1
```

richtet sich beispielsweise an sämtliche Dateien einer Diskette, die mit den Buchstaben »TE« beginnen. Befinden sich vielleicht die Programme »TEST« und »TEXT« auf der eingelegten Diskette, werden beide gelöscht, und bei der Statusmeldung »FILES SCRATCHED« die Zahl 02 (2 Dateien) ausgegeben.

Kommen wir aber nun zu den Meldungen, die einen richtigen Fehler anzeigen. Jede besitzt eine Nummer zwischen 0 und 74. Doch nicht jeder Nummer ist einer Fehlermeldung zugeordnet (es gibt »nur« 34). Wir werden diese im folgenden auch nicht nach Nummern sortiert vorstellen, sondern wir gruppieren sie nach ihrer Funktion und Bedeutung.

Beginnen wir gleich mit einigen einfachen Fehlern, die Sie bei der Arbeit mit der Floppy 1541 relativ häufig erhalten werden. Sie sind jedoch vergleichsweise harmlos.

Es kann vorkommen, daß Sie versehentlich mit dem Diskettenlaufwerk arbeiten wollen, obwohl sich keine Diskette darin befindet. Die Floppy 1541 bemerkt dies recht schnell und antwortet sofort:

```
74,DRIVE NOT READY,00,00
```

Sollten Sie aber dennoch eine Diskette in den Laufwerkschacht geschoben haben, kann die Ursache an einer unformatierten Diskette oder an Problemen bei der Schreib-/Lesekopfbewegung liegen. Im ersten Fall muß die Diskette formatiert werden (siehe Handbuch). Im zweiten Fall hilft manchmal die nachstehende Befehlsfolge:

```
OPEN 1,8,15
PRINT#1,"I"
CLOSE 1
```

Befindet sich unter Ihrer Diskettensammlung auch eine Diskette, die auf einer anderen Commodore-Floppystation (zum Beispiel CBM 3040) beschrieben wurde, kann die Floppy 1541 sie unter Umständen nur lesen. Beim Schreiben gibt es Schwierigkeiten; die Diskette wird sofort als fremd erkannt. Eine uns schon bekannte Meldung wird über den Kommandokanal gesendet:

```
73,CBM DOS V2.6 1541,00,00
```

Nun blinkt die rote LED und zeigt uns einen Fehler an. Die Nachricht soll uns daran erinnern, daß wir nur Disketten verwenden dürfen, die von einer Floppy 1541 beschrieben wurden oder zu diesem Diskettensystem passen.

Endlich liegt die richtige Diskette im Laufwerk. Voller Stolz wollen Sie ein selbstgeschriebenes Basic-Programm mit SAVE speichern. Doch schon nach kurzer Zeit blinkt erneut die rote Leuchtdiode. Auf dem Fehlerkanal wartet folgende Meldung auf ihren Abruf:

```
26,WRITE PROTECT ON,00,00
```

Hier ist offensichtlich die Schreibschutzkerbe der Diskette (die kleine eckige Aussparung an der Seite des Diskettenmantels) mit einer Plakette überdeckt. Wie der Name schon sagt, kann man durch Überkleben dieser Kerbe eine Diskette vor dem Überschreiben oder Löschen schützen. Entfernt man die Abdeckung wieder, ist die Speicherung wieder möglich.

Nummer	Fehler	Bedeutung
00	OK	Befehl wurde ordnungsgemäß ausgeführt
01	FILES SCRATCHED,XX	XX Dateien wurden gelöscht
26	WRITE PROTECT ON	Diskette ist durch eine Schreibschutzplakette vor dem Überschreiben geschützt
72	DISK FULL	Diskette ist voll, oder es wurden bereits 144 Dateien gespeichert
73	CBM DOS V2.6 1541	Einschaltmeldung; Diskette mit Fremdformat eingelegt

Tabelle 2. Ein Teil der Fehlermeldungen der 1541

»SYNTAX ERROR« gleich sechsmal

Haben Sie den Aufkleber von der Kerbe entfernt, können Sie mehrere Programme mit SAVE auf die Diskette schreiben. Irgendwann ist jedoch die Kapazität der Diskette erschöpft. Paßt ein Programm nicht mehr darauf, wird die Floppy 1541 die Nachricht

```
72,DISK FULL,00,00
```

auf dem Fehlerkanal bereitstellen. Dieser Fehler tritt auch auf, wenn Sie zu viele verschiedene Programme speichern wollen. Auf einer Diskette können nämlich maximal 144 Dateien abgelegt werden, selbst wenn ansonsten noch Speicherplatz zur Verfügung steht.

Betrachten Sie Tabelle 2, dann sehen Sie alle bisher besprochenen Status- und Fehlermeldungen. Eine Kurzbeschreibung gibt nochmals einen Überblick über deren Bedeutungen und Ursachen.

Kommen wir nun zu einem Fehlertyp, den Sie wahrscheinlich von Ihrem C64 kennen: dem »SYNTAX ERROR«. Er erscheint auf Ihrem Computer beispielsweise, wenn Sie einen Befehl eingetippt haben, den der C64 nicht kennt. Eine andere Ursache kann sein, daß eine Anweisung falsch, das heißt, nicht den Regeln entsprechend, eingegeben wurde. Der Computer beschwert sich mit einem »SYNTAX ERROR«, da die Syntax (Schriftform) eines Befehles nicht ordnungsgemäß befolgt wurde. Ähnlich verhält es sich auch bei der Floppy 1541. In vorangegangenen Abschnitten haben wir erwähnt, daß die Floppystation verschiedene Kommandos verarbeiten kann. Wie beim C64 muß bei der Eingabe, also dem Senden über den Kommandokanal eine Syntax eingehalten werden. Wird diese ignoriert, antwortet das Diskettenlaufwerk mit einem »SYNTAX ERROR«. Doch macht die Floppy 1541 bei der Art der Fehlerkennung genaue Unterschiede. Sie kennt insgesamt sechs Syntax-Fehlermeldungen mit den Fehlernummern 30 bis 34 und 39. Einige davon haben jedoch sehr ähnliche Bedeutungen.

Nummer	Fehler	Bedeutung
30	SYNTAX ERROR	allgemeiner Syntaxfehler
31	SYNTAX ERROR	ungültiger Befehl
32	SYNTAX ERROR	Befehlszeile zu lang
33	SYNTAX ERROR	unerlaubte Verwendung eines Jokerzeichens
34	SYNTAX ERROR	Dateiname ist nicht angegeben
39	SYNTAX ERROR	ungültiger Befehl

Tabelle 3. Die Floppy 1541 kennt sechs »SYNTAX ERROR«

Ein »SYNTAX ERROR« liegt vor, wenn ein Kommando, das über den Kommandokanal gegeben wird, nicht als Befehl identifiziert werden kann. Oftmals sind falsch angeordnete Parameter, die mit dem Befehl gesendet werden müssen, die Übeltäter. In diesem Fall wird die Floppystation einen Syntax-Error mit der Nummer 30 ausgeben. Existiert der Befehl jedoch gar nicht, beschwert sich das Laufwerk mit einem Fehler der Nummer 31. Ist man aber fest davon überzeugt, einen korrekten Befehl eingegeben zu haben, kann der Grund bei einem Leerzeichen liegen, das man versehentlich vor das Kommandowort gesetzt hat. Das Diskettenlaufwerk akzeptiert nur Anweisungen, die an erster Position stehen. Die dritte Meldung, die auf einen falschen Befehl hinweist, ist der Syntax-Error mit der Nummer 39. Er tritt auf, wenn ein Befehl von der Floppy 1541 nicht eindeutig interpretiert werden kann, weil er zum Beispiel zusammen mit anderen Zeichen in einer Kommandozeile steht.

Eine sogenannte Befehlszeile, das heißt die Befehlsfolge, die über den Kommandokanal geschickt wird, darf bei der Floppy 1541 maximal 58 Zeichen lang sein. Bei den meisten DOS-Anweisungen wird man dieses Maximum nicht überschreiten, da die Anordnung der eventuell mitgesendeten Parameter fest vorgeschrieben ist. Einigen Kommandos können jedoch variable Parameterlisten angehängt werden. In einem solchen Fall kann es leicht geschehen, daß die Befehlszeile länger als 58 Zeichen wird. Ein »32, SYNTAX ERROR,00,00« ist die Folge.

Kennt man die wichtigsten Diskettenbefehle, lernt man schnell, welche davon einen oder mehrere Dateinamen benötigen. Es ist dann höchst unwahrscheinlich, daß man zum Beispiel beim Löschen einer Datei vergißt, den Namen der betreffenden Datei anzugeben. Doch das Fehlen eines einzigen Zeichens kann es für das DOS der 1541 unmöglich machen, die Bezeichnung zu erkennen. Ein kleines Beispiel soll dies demonstrieren:

```
OPEN 1,8,15
PRINT#1,"STEST"
CLOSE 1
```

Wie Sie sehen können, fehlt hier der Doppelpunkt zwischen Befehl (S) und Dateiname (TEST). In diesem Fall kann die Floppy 1541 den Namen »TEST« nicht entziffern und gibt einen besonderen Fehler mit der Nummer 34 aus. Er weist uns darauf hin, daß ein Dateiname in der Kommandoanweisung fehlt, der für die Bearbeitung des angegebenen Befehls notwendig ist.

Der sechste »SYNTAX ERROR« steht ebenfalls mit Dateinamen im Zusammenhang. Seine Nummer ist 33. Tritt dieser Fehler auf, wissen wir, daß wir eines der Jokerzeichen falsch angewendet haben. Die Joker (? und *) sind bekanntlich Hilfsmittel, um Dateinamen abzukürzen, oder wie bei dem SCRATCH-Kommando, um mehrere Dateien einer Diskette gleichzeitig anzusprechen. Bei einigen Befehlen, die Namen als Parameter benötigen, dürfen die Jokerzeichen nicht eingesetzt werden. Eines dieser Kommandos ist Ihnen geläufig. Es ist das NEW- oder N-Kommando zum Formatieren einer Diskette. Mißbrauchen wir nun einmal vorsätzlich den Joker »*«:

```
OPEN 1,8,15
PRINT#1,"N:TESTDIS*,TD"
CLOSE 1
```

Die Diskettenstation wird dies nicht akzeptieren und folgende Fehlermeldung erzeugen:

```
33, SYNTAX ERROR,00,00
```

Womit wir den letzten der besprochenen Syntax-Fehler provoziert hätten. In Tabelle 3 haben wir alle Syntaxfehler mit kurzen Erläuterungen zusammengefaßt.

Neben Programmen kann die Floppy 1541 auch andere Daten auf einer Diskette speichern. Wenn Sie zum Beispiel ein Programm geschrieben haben, das die Adressen Ihrer Freunde verwaltet, sollten deren Daten für längere Zeit gespeichert werden, denn der Speicherinhalt des C64 wird beim Ausschalten gelöscht. Hierfür bietet die 1541 eine besondere Lösung: die sequentielle Datenspeicherung. Die Diskettenstation erlaubt es Ihnen, beliebige Daten in sequentiellen Dateien (SEQ-Dateien) abzulegen, um sie bei Bedarf jederzeit wieder in den Speicher des Computers zurückzuholen. Die Daten werden dabei der Reihe nach (sequentiell) auf die Diskette geschrieben und können ebenso wieder gelesen werden. Die Handhabung ist denkbar einfach. Dennoch können dem Programmierer einige Mißgeschicke unterlaufen.

Stolpersteine bei der Dateiverwaltung

Um die Fehlermeldungen, die die Floppy 1541 als Hilfestellung bereitstellt, zu erläutern, werden wir nun eine sequentielle Datei eröffnen. Selbst wenn Sie noch keine Erfahrung mit der Datenspeicherung auf der Floppy 1541 gemacht haben, werden Sie den Gedankengängen leicht folgen können, da wir die Erklärungen so allgemein wie möglich formulieren, ohne speziell auf Befehlsstrukturen einzugehen.

Die Handhabung einer Datei gliedert sich in vier Schritte. Sie erfolgt auf ähnliche Weise wie die Bedienung des Kommandokanals:

1. Öffnen der Datei mit OPEN
2. Schreiben von Daten mit PRINT#
3. oder Lesen von Daten mit INPUT# oder GET#
4. Schließen der Datei mit CLOSE

Nachdem wir uns einen Namen für unsere Datei ausgesucht haben (er soll »TESTDATEI« lauten), kann die Datenspeicherung beginnen. In unserer Zerstreutheit übergehen wir Punkt 1 der Liste und versuchen sofort, Daten auf die Diskette zu schreiben. Dieser äußerst gravierende Mangel fördert schon die erste Fehlermeldung zutage. Ohne daß der Fehlerkanal der Floppystation nach einer Fehlermel-

dung abgerufen werden muß, beschwert sich der Computer mit einem »?FILE NOT OPEN ERROR«, da wir die Datei nicht mit OPEN geöffnet haben. Der Fehlerkanal der Floppy 1541 zeigt das gleiche Resultat:

```
61, FILE NOT OPEN,00,00
```

Die Datei muß zunächst unter einem Namen eröffnet werden. Unglücklicherweise befindet sich auf unserer Diskette bereits eine andere Datei mit dem gleichen Namen. Die Organisation der Diskette erlaubt jedoch zwei Dateien gleichen Namens nicht, und so werden wir mit dem nächsten Fehler konfrontiert. Die Floppystation meldet:

```
63, FILE EXISTS,00,00
```

Ein neu gewählter Name (zum Beispiel »TESTFILE«) löst dieses Problem. Die gewünschten Daten können nach all den anfänglichen Schwierigkeiten endlich auf die Diskette geschrieben werden.

Hat man die besagten Daten in einer sequentiellen Datei abgelegt, ist es besonders wichtig, den Abschluß der Arbeit mit dem Befehl CLOSE anzukündigen. Die Datei muß wieder geschlossen werden. Ungeachtet dieser Tatsache schalten wir jedoch unseren Computer ab, ohne den letzten notwendigen Schritt zu tun. Die Datei wird damit nicht ordnungsgemäß geschlossen.

Fehler des Schicksals

Schließlich wollen wir die Daten wieder in den Computer lesen. Versehentlich legen wir eine andere Diskette in das Laufwerk, die unsere Datei nicht enthält. Beim Öffnen der Datei kann die Floppystation die gesuchten Daten nicht finden und antwortet mit einer Fehlermeldung:

```
62, FILE NOT FOUND,00,00
```

Die Datei wurde nicht gefunden. Parallel dazu gibt der C 64 ebenfalls die gleiche Meldung aus:

```
?FILE NOT FOUND ERROR
```

Auch beim Laden von Programmen kann dieser Fehler auftreten, denn Programme sind eine besondere Art von Dateien (PRG-Dateien). Haben wir nach längerem Suchen endlich die richtige Diskette in das Laufwerk eingeschoben, können wir einen erneuten Versuch wagen. Leider geben wir dabei den falschen Filetyp an, denn die Floppy 1541 kennt neben den Programmen (PRG) und den sequentiellen (SEQ) Dateien noch weitere Arten der Datenspeicherung. So kann man auch eine »User«-Datei (USR) eröffnen, die sich in der Handhabung jedoch nicht von SEQ-Dateien unterscheidet, und es gibt noch die relative Datenspeicherung (REL). Wir werden gleich auf sie eingehen.

Jedes Mal, wenn wir eine Datei öffnen, um Daten zu schreiben oder zu lesen, müssen wir den entsprechenden Dateityp angeben. Sie erinnern sich, daß unsere Datei vom Typ SEQ war. Wollen wir wieder darauf zugreifen, ist dieser Typ wiederholt anzugeben. Geschieht dies nicht, reagiert das Floppylaufwerk mit einem Fehler. Er kann provoziert werden, wenn wir unsere Datei mit dem Namen »TESTFILE« nicht als SEQ- sondern beispielsweise als USR-Datei ansprechen wollen. Die Antwort des Laufwerks folgt nach kurzer Zeit:

```
64, FILE TYPE MISMATCH,00,00
```

Doch nun wollen wir unsere Datei ordnungsgemäß mit richtigem Filetyp und Namen zum Lesen von Daten öffnen. Obwohl wir alle Bedingungen beachtet haben, beginnt die rote Leuchtdiode erneut zu blinken. Eine weitere Meldung wartet auf dem Fehlerkanal:

```
60, WRITE FILE OPEN,00,00
```

Mit Schrecken erinnern wir uns, daß wir beim Anlegen der Datei das Schließen mit CLOSE vergessen haben. Sie wurde damit nicht ordnungsgemäß abgeschlossen und kann jetzt nicht mehr gelesen werden. Die obige Nachricht weist uns unmißverständlich darauf hin. Die ganze Mühe, die wir uns mit unserer Datei machten, war also umsonst. Mit viel Aufwand und guten Programmierkenntnissen kann man zwar eine Datei nachträglich schließen und auf diese Weise die verloren geglaubten Daten retten, doch ist dies ziemlich zeitaufwendig.

Es ist sehr unwahrscheinlich, daß man all die Torturen, die wir eben spielerisch demonstriert haben, bei der Arbeit mit einer Datei in dieser Weise durchleben muß. Denn hat man sich etwas mit der Dateibedienung der Floppy 1541 beschäftigt, kann man viele dieser Fehler von Beginn an vermeiden.

Wir sind aber bei der Erläuterung der Fehlermeldungen zur Bearbeitung von Dateien noch nicht am Ende. In einem früheren Abschnitt dieses Artikels haben Sie kennengelernt, daß es neben der sequentiellen auch die relative Datenspeicherung gibt. Sie wird verwendet, wenn man schnell auf bestimmte Elemente der Daten zugreifen will. Diese Art der Datenspeicherung ist aber wesentlich komplizierter als etwa die sequentielle. Jedes Datenelement hat bei relativen Dateien eine begrenzte Länge und wird als Datensatz oder »Record« bezeichnet. Eine Datei kann nun viele dieser Datensätze enthalten, die alle aufsteigend numeriert werden. Hat man nun alle gewünschten Datensätze auf der Diskette gespeichert, kann man durch Angabe der Nummer den entsprechenden Datensatz direkt ansprechen, ohne etwa davor befindliche Daten überlesen zu müssen.

Hierbei können besondere Fehler auftreten, die die Floppy 1541 auch mit besonderen Meldungen bedacht hat. Die Nummern dazu lauten 50, 51 und 52.

Es kann beispielsweise vorkommen, daß man auf einen nicht vorhandenen Datensatz zugreifen will. Die Floppystation übermittelt uns dann die folgende Nachricht:


```
50, RECORD NOT PRESENT,00,00
```

Wir wissen somit, daß sich der Datensatz der angegebenen Nummer nicht in der Datei befindet. Schreibt man nun aber dennoch Daten hinein, ist die Floppy 1541 sehr hilfsbereit und erweitert die Datei um diesen zusätzlichen Datensatz. Beim Speichern von Datensätzen muß man jedoch sehr vorsichtig sein. Es wurde bereits erwähnt, daß alle Datensätze nur eine bestimmte Länge (Anzahl von Zeichen) haben dürfen. Ignoriert man diese Tatsache, sieht man sich mit einem weiteren Fehler konfrontiert:

```
51, OVERFLOW IN RECORD,00,00
```

Eine relative Datei kann jederzeit um beliebige Records erweitert werden. Irgendwann reicht aber die Speicherkapazität der verwendeten Diskette nicht mehr aus, um neue Datensätze aufzunehmen. Wenn es soweit ist, teilt uns die Floppystation dies durch eine Fehlermeldung mit:

```
52, FILE TOO LARGE,00,00
```

Die Datei kann nicht mehr erweitert werden, da sie die, für relative Dateien vorgeschriebene, Maximallänge er-

Nummer	Fehler	Bedeutung
60	WRITE FILE OPEN	Zugriff auf eine nicht geschlossene Datei
61	FILE NOT OPEN	Datei ist nicht geöffnet
62	FILE NOT FOUND	Datei wurde nicht gefunden
63	FILE EXISTS	Datei mit angegebenem Namen existiert bereits
64	FILE TYPE MISMATCH	falscher Dateityp angegeben
Nur bei relativen Dateien		
50	RECORD NOT PRESENT	Datensatz der angegebenen Nummer existiert nicht
51	OVERFLOW IN RECORD	Datensatz zu lang
52	FILE TOO LARGE	Datei kann nicht erweitert werden, da Diskette voll

Tabelle 4. Die möglichen Fehler bei der Arbeit mit Dateien

reicht hat. Diese Meldung gilt aber nur bei relativen Dateien und ist nicht zu verwechseln mit der »72, DISK FULL,00,00«-Nachricht, deren Bedeutung Sie schon kennengelernt haben.

Damit ist das Reservoir an Fehlermeldungen bezüglich der Dateibehandlung erschöpft. Sollte Ihnen bei der Fülle an Informationen schwindlig geworden sein, blicken Sie bitte auf Tabelle 4. Hier haben wir alle die in diesem Abschnitt besprochenen Fehlermeldungen noch einmal zusammengestellt.

Alle Fehlermeldungen, die wir bisher bei unserer Exkursion durch die Welt der »Errors« erforscht haben, werden Ihnen im Laufe der Zeit oftmals begegnen, wenn Sie nur intensiv genug mit der Floppy 1541 arbeiten. Des weiteren zeichnen sie sich durch eine Gemeinsamkeit aus: Sie alle sind Anwenderfehler, das heißt Fehler, die auf das Unwissen oder die Unzulänglichkeit des Programmierers zurückzuführen sind. Hat man sich genügend Wissen über die Floppystation angeeignet, wird man sie größtenteils vermeiden können.

Auf unserem Pfad, der uns jetzt weg von Programmen und Dateien in das Innere des Diskettenlaufwerks führt, werden wir jedoch Fehlermeldungen kennenlernen, deren Ursache meist nicht durch eine Fehlbedienung gegeben ist. Hier sind oft minderwertige Disketten mit beschädigter Magnetoberfläche oder fremde Umwelteinflüsse die Übeltäter, die das ordnungsgemäße Speichern von Daten behindern und so für Fehler sorgen.

Hier ein Beispiel aus dem täglichen Leben: Sie laden eines Ihrer beliebtesten Spielprogramme und nehmen schon den Joystick zur Hand. Doch plötzlich gibt die Floppystation seltsame Geräusche von sich, während die rote Leuchtdiode, die zuvor konstant leuchtete, unregelmäßig zu flackern beginnt. Schließlich ertönt ein scheußliches Rattern und das Laufwerk hält mit einer Fehlermeldung an, obwohl man alle Eingaben in den Computer korrekt vorgenommen hat. Das Spielprogramm kann nicht mehr geladen werden und scheint für immer verloren. Wir werden diese Art von Fehlern im folgenden besprechen.

Warum Formatieren?

Um ihre Bedeutung genau zu verstehen, müssen wir uns zunächst mit der Organisation einer Diskette bei der Floppy 1541 beschäftigen. Wir werden kennenlernen, auf welche Weise unser Diskettenlaufwerk Programme und andere Daten auf einer Diskette ablegt.

Wenn mehrere Programme auf einer Kassette gespeichert werden sollen, läuft dies nach einem einfachen Schema ab. Die Daten werden der Reihe nach auf das Magnetband geschrieben. Ist ein Programm zu Ende, kann gleich dahinter ein neues Programm gespeichert werden. Man kann dies fortführen, bis die Kassette abgelaufen ist. Durch Abspielen des gesamten Bandes oder manuelles Vor- und Zurückspulen läßt sich nach einem bestimmten Programm suchen, was allerdings sehr lange dauern kann.

Eine Diskette kann ebenfalls mehrere Programme enthalten. Das einfache Prinzip der Kassettenspeicherung ist aber auf der runden Magnetscheibe einer Diskette kaum zu verwirklichen. Des weiteren zeichnet sich ein Floppylaufwerk durch schnelle Zugriffszeiten auf alle Daten aus; das heißt ein Programm, das irgendwo auf der Diskette abgelegt ist, kann durch Angabe seines Namens sofort gefunden und in den Speicher des Computers geladen werden. Dies bedarf einer geregelten Organisation, die es der Floppystation erlaubt, sich schnell und problemlos auf der Diskette zurechtzufinden. Zu diesem Zweck müssen sich auf der Magnetoberfläche Markierungen befinden, die dem Diskettenlaufwerk bei der Orientierung behilflich sind. Sie sind von Gerät zu Gerät verschieden und bestimmen das Format einer Diskette. Um die für die Floppy 1541 notwendigen Markierungen auf eine Diskette zu bringen, muß diese formatiert werden. Vielleicht wissen Sie, daß die Floppystation einige Zeit dazu benötigt (etwa 90 Sekunden). Sehen wir uns einmal kurz an, was dabei mit der Diskette geschieht.

Die »Innereien einer Diskette

Beim Formatieren wird die Magnetscheibe der Diskette in 35 (magnetische) Spuren aufgeteilt, die konzentrisch um das Mittelloch angeordnet sind. Sie erhalten von außen nach innen die Nummern 1 bis 35. Während sich die Diskette dreht, kann der Schreib-/Lesekopf, von einem kleinen Motor geführt, auf jede dieser Spuren gelangen, um dort Daten zu schreiben oder zu lesen. Da eine solche Spur (englisch: Track) als Speichereinheit noch zu groß ist, um bequem bearbeitet werden zu können, wird sie in noch kleinere Untereinheiten zerlegt: die Sektoren. Ihre Anzahl kann von Spur zu Spur unterschiedlich sein, da deren Länge auf der Diskette von außen nach innen abnimmt. Pro Spur werden die Sektoren ab Null aufsteigend numeriert. Jeder Sektor kann somit durch eine Spur- und Sektornummer eindeutig bestimmt werden.

Auch innerhalb eines Sektors herrscht eine strenge Ordnung. Nach einem Vorspann (Kopf oder auch »Header«), der wichtige Daten über den entsprechenden Sektor enthält, folgt der Datenblock. Hier werden nun endlich die Daten abgelegt. Einen ausführlichen Kurs darüber finden Sie übrigens in unserem Floppy-Sonderheft 9/87. Jeder Sektor kann 254 Byte aufnehmen. Da ein Programm in der Regel länger ist, wird es beim Speichern auf mehrere Sektoren verteilt.

Anschließend wird beim Formatieren auf der Diskette ein Inhaltsverzeichnis angelegt, das die Namen der zukünftig gespeicherten Dateien aufnehmen wird. Daneben wird es wichtige Daten enthalten, die der Floppystation bei der Suche nach einer bestimmten Datei behilflich sind. Das Directory, wie das Inhaltsverzeichnis einer Diskette auch genannt wird, benötigt die gesamte Spur 18.

Man wird verstehen, daß bei all der komplizierten Organisation bereits ein geringer Fehler (oft genügt ein falsches Bit) größte Probleme ergeben kann. Wie bei der Speicherung auf der Datasette trifft die Floppy 1541 deshalb einige Vorkehrungen, um eventuelle Fehler möglichst zu vermeiden.

Wird zum Beispiel infolge eines SAVE-Befehles vom Computer ein Sektor auf einer Diskette mit Daten beschrieben, wird sogleich automatisch überprüft, ob alle Bytes ordnungsgemäß gespeichert wurden (VERIFY). Daneben werden, ähnlich wie bei der Kassettenspeicherung, Prüfsummen über die entsprechenden Daten erstellt und bei einem Lesezugriff mit den geladenen Bytes verglichen. Man findet jeweils eine Prüfsumme im Kopf und im Datenblock eines Sektors.

Man sieht, daß für die Datensicherheit gut gesorgt ist. Insbesondere das automatische, floppyinterne VERIFY beim Schreiben eines jeden Sektors verhindert, daß die Daten bereits fehlerhaft geschrieben werden. Doch trotz all dieser Vorkehrungen können Probleme bei der Datenspeicherung auf einer Diskette entstehen.

Obgleich die Ursachen eines Fehlers meist die gleichen sind (zum Beispiel fehlerhafte Disketten oder fremde Umwelteinflüsse), können sie in den verschiedensten Bereichen auftreten, die die Floppy 1541 sorgsam zu unterscheiden weiß. Man faßt sie allgemein unter dem Begriff »Lese-

fehler« zusammen, da durch irgendwelche Leseprobleme einer Diskette gewisse Daten oder Markierungen nicht korrekt identifiziert werden können. Die Floppy 1541 kennt insgesamt 9 dieser Lesefehler, von denen 6 die allgemeine Bezeichnung »READ ERROR« haben. Sie treten nur beim Lesen von Daten auf. Zwei Fehlermeldungen tragen den Namen »WRITE ERROR«, da sie nur beim Schreiben von Daten, wie zum Beispiel bei SAVE, in Erscheinung treten. Wir haben sie in sortierter Reihenfolge in Tabelle 5 abgedruckt. Dort finden Sie auch den neunten Lesefehler mit dem Namen »DISK ID MISMATCH ERROR«.

Diese Gruppe von Fehlern läßt sich in zwei Typen unterteilen: die »Soft« und die »Hard«-Errors, was Sie ebenfalls der Tabelle entnehmen können.

Obwohl alle Lesefehler größte Probleme bereiten, sind die Soft-Errors (»weiche« Fehler) in unserer Liste etwas harmloser als die Hard-Errors (»harte« Fehler). In den meisten Fällen können bei Fehlern dieses Typs die Daten des betroffenen Sektors noch relativ einfach und ohne gravierende Verluste gerettet werden. Ein »READ ERROR« der Nummer 22 weist beispielsweise darauf hin, daß der Header eines Sektors zwar einwandfrei gelesen werden konnte, die Floppystation aber den normalerweise folgenden Datenblock nicht gefunden hat. Die Meldung kann zum Beispiel lauten:

```
22, READ ERROR,14,09
```

Meist hat hier eine wichtige Markierung im Datenblock nicht den richtigen Wert, der Datenblock selbst ist aber korrekt. Mit etwas Programmieraufwand kann der Sektor noch gerettet werden, was uns an dieser Stelle aber nicht genauer interessieren soll.

Der »Fehlerteufel« schlägt zu

Bei den Fehlermeldungen bekommen nun auch die beiden Zahlen hinter dem Klartext, die bisher meistens auf Null standen, eine Bedeutung. Sie geben die Spur- und Sektornummer des Blocks an, in dem der Fehler aufgetreten ist. In unserem Beispiel ist es der Sektor 9 auf Spur 14. Wir werden diesen Sektor ab jetzt weiterhin als Beispiel verwenden.

Meldet die Floppystation

```
23, READ ERROR,14,09
```

handelt es sich ebenfalls um einen »Soft-Error«. Eine falsche Prüfsumme war hier die Ursache. Wie Sie schon wissen, enthält ein Sektor zwei Prüfsummen. Hier ist die Prüfsumme des Datenblocks gemeint, die mit der der gelesenen Datenbytes nicht übereinstimmt. In diesem Fall besteht die Hoffnung, daß lediglich die Prüfsumme falsch ist, die Daten jedoch korrekt geschrieben wurden. Einige Programme, die sich mit der Diskettenbehandlung beschäftigen, wie zum Beispiel bestimmte Kopierprogramme, erlauben es deshalb, einen Sektor trotz eines »23, READ ERROR« zu lesen, um die Prüfsumme anschließend zu korrigieren. Meist sind jedoch fehlerhafte Datenbytes die Sünder. Die eben beschriebene Korrektur bleibt dann ohne Erfolg.

Ein Prüfsummenfehler im Sektor-Kopf kann ohne großen Aufwand nicht mehr repariert werden. Er hat einen harten Lesefehler zur Folge:

```
27, READ ERROR,14,09
```

Ein weiterer »Soft-Error« ist vom Typ »WRITE ERROR«. Er tritt bereits beim Schreiben von Daten auf. Wurde bei der Vergleichskontrolle eines eben gespeicherten Sektors ein Fehler in den Datenbytes entdeckt, meldet die Floppystation:

```
25, WRITE ERROR,14,09
```

Oftmals liegt dieser Meldung eine fehlerhafte Magnetbeschichtung der Diskette zugrunde. Doch ist der Fehler nicht besonders tragisch, da sich die Daten, zum Beispiel ein Programm, in der Regel noch im Speicher des Computers befinden. Man kann also einen weiteren Speicherversuch auf einer anderen Diskette vornehmen. Hat man auf diese Weise eine »Problemdiskette« erkannt, ist es ratsam, die restlichen Dateien von dieser Diskette auf einen neuen Datenträger zu kopieren. Vorsicht ist in solchen Fällen besser als der Verlust sämtlicher Daten auf der schadhaften Diskette.

Harte Probleme mit »harten« Fehlern

Bei den »Soft-Errors« ist der Header (Kopf) eines Sektors in der Regel noch in Ordnung. Der Sektor kann auf der Diskette also noch gefunden werden. Handelt es sich jedoch um einen »Hard-Error«, ist schon der Vorspann des betreffenden Sektors nicht mehr lesbar, wie es beispielsweise der Fehler mit der Nummer 20 anzeigt:

```
20, READ ERROR,14,09
```

Er besagt, daß die Floppy 1541 den Kopf eines Sektors nicht ausfindig machen konnte. Der Fehler kann sich dabei entweder auf nur einen Sektor oder eine ganze Spur beziehen. Ist die gesamte Spur betroffen, sind die darauf befindlichen Daten normalerweise für immer verloren, wie es auch beim »21, READ ERROR« der Fall ist. Hier ist es der Floppystation unmöglich, eine für das Lesen und Schreiben wichtige Markierung zu finden. Auch bei diesem Fehler braucht man sich keine Hoffnung auf eine Rettung der Daten machen.

Nummer	Fehler	Typ	Bedeutung
20	READ ERROR	hard	Blockheader nicht gefunden
21	READ ERROR	hard	Sync-Markierung nicht gefunden
22	READ ERROR	soft	Datenblock nicht gefunden
23	READ ERROR	soft	Prüfsummenfehler in Datenblock
24	READ ERROR	hard	Fehler bei der GCR-Recodierung
25	WRITE ERROR	soft	Fehler beim Verifizieren
27	READ ERROR	hard	Prüfsummenfehler im Sektor-Header
28	WRITE ERROR	hard	Block zu lang; nächster Block wurde überschrieben (Hardware-Defekt)
29	DISK ID MISMATCH	hard	falsche ID im Sektor-Header

Tabelle 5. Lesefehler – der Schrecken aller Programmierer. Nicht immer liegt es an einer defekten Diskette.

Ein

```
29, DISK ID MISMATCH,14,09
```

tritt auf, wenn die »ID« (»IDentifizierung«) eines Sektors nicht mit der ID der Diskette übereinstimmt, die Sie beim Formatieren angegeben hatten. Bei diesem Fehler ist meist ein zerstörter Sektor-Header die Ursache.

Relativ selten ist der Lesefehler mit der Nummer 24 zu finden. Er bezieht sich auf die Codierung der Daten, bevor sie in einen Sektor auf der Diskette geschrieben werden. Aus organisatorischen Gründen werden die Daten nicht im Originalzustand gespeichert, sondern zuvor in einer gewissen Weise codiert. Nach dem Lesen müssen sie selbstverständlich wieder recodiert, das heißt zurückübersetzt werden. Im Normalfall entstehen hierbei keine Probleme. Im Fehlerfall können Werte auftreten, die nicht entziffert werden können. Die Antwort der Floppy 1541 lautet dann:

```
24, READ ERROR,14,09
```

Ebenso rar wie der Fehler 24 ist der zweite »WRITE ERROR« in der Tabelle 5. Er hat die Nummer 28. Hier wurde ein Sektor von einem davor befindlichen einfach überschrieben, so daß die Floppystation die Anfangsmarkierun-

gen dieses Sektors nicht mehr finden kann. Dieser Fehler kann auftreten, wenn sich die Diskette zu schnell dreht, weil die Geschwindigkeitsregelung des Laufwerkmotors ausgefallen ist. Die Sektoren werden dann zu »lang« und überschreiben sich gegenseitig.

Nummer	Fehler	Bedeutung
65	NO BLOCK	Block bereits belegt
66	ILLEGAL TRACK OR SECTOR	Zugriff auf ungültige Spur oder Sektor
67	ILLEGAL TRACK OR SECTOR	Zeigerbytes in Datenblock zeigen auf ungültige Spur oder Sektor
70	NO CHANNEL	Kanal bereits belegt
71	DIR ERROR	zerstörte BAM

Tabelle 6. Außenseiter: Diese Lesefehler passen in keine der vier Gruppen

Bleiben zu guter Letzt noch einige Fehlermeldungen, die nicht so recht in eine unserer Gruppen passen wollen. Drei dieser Fehlermeldungen werden bei der normalen Anwendung der Floppy 1541 wohl nicht auftreten. Der fortgeschrittene Programmierer, der einige Direkteingriffe auf einer Diskette vornimmt, wird Ihnen öfter begegnen. Das sind die Fehler mit den Nummern 65, 66 und 71, die wir hier nicht näher erläutern wollen. Sie sind aber in unserer Tabelle 6 dennoch aufgelistet. Lediglich der Fehler der Nummer 67 kann häufiger auftreten. Die Meldung lautet folgendermaßen:

```
67, ILLEGAL TRACK OR SECTOR,14,09
```

Da ein Programm meist länger ist als 254 Byte, muß es auf mehrere Sektoren verteilt werden. Damit die Floppystation nach dem Lesen eines Sektors weiß, in welchem Folgesektor das Programm oder die Datei fortgesetzt wird, enthalten die ersten zwei Byte des Datenblocks eines Sektors stets die Spur- und Sektornummer des darauffolgenden Programmteils. Durch unsachgemäßen Zugriff auf diesen Sektor ist es möglich, daß dieser Zeiger auf einen Sektor zeigt, der überhaupt nicht existiert. Spur 75 ist beispielsweise bei der Floppy 1541 nicht vorhanden, da die höchste Spur die Nummer 35 besitzt. Ein »ILLEGAL TRACK OR SECTOR«-Fehler entsteht. Wichtig ist dabei noch zu wissen, daß auch eine unerlaubte Sektornummer diese Fehlermeldung zur Frage hat. Hier müssen zur Unterscheidung die Spur- und Sektorangaben beachtet werden.

An dieser Stelle ist es bestimmt kein »Fehler«, das Thema der Fehlermeldungen zu beenden. Manche Bereiche, die wir in diesem Bericht angeschnitten haben, werden sicherlich Fragen offenlassen. Zu ihrer Beantwortung weisen wir auf einschlägige Literatur wie »Die Floppy 1541« von Markt&Technik hin, die sich wesentlich intensiver mit diesem oder jenem Problem der Floppy 1541 auseinandersetzen kann. Die große Anzahl an Fehlermeldungen läßt die Komplexität der 1541 nur erahnen. Vielleicht ist das aber ein Ansporn, sich näher mit diesem interessanten Peripheriegerät zu beschäftigen. (Michael Thomas/ef)

Floppyfehler abfangen

Manchem Programmierer sträuben sich die Haare, wenn er das Wort »Fehlermeldung« hört. Manchmal kann diese Einrichtung des Computers jedoch sehr sinnvoll angewendet werden.

Tritt während der Programmausführung ein Fehler auf, so bricht der Computer mit einer mehr oder weniger deutlichen Fehlermeldung ab. Für Fehler beim Zugriff auf die Diskette gilt dies jedoch nicht. Hier blinkt lediglich eine Leuchtdiode am Laufwerk, und das Programm läuft oft weiter. Dies können wir für ein programmiertes Abfangen solcher Fehler ausnutzen. Allerdings benötigen wir dazu mehr Informationen. Das Floppylaufwerk hält diese für uns bereit - wir holen sie mit der Anweisung

```
10 OPEN 14,8,15:INPUT#14,A,B$,C,D:CLOSE 14
```

ab. Die Zeile geben Sie bitte mit der Zeilennummer ein. Nach Eingabe von RUN wird die Fehlermeldung auf dem Bildschirm ausgegeben. Das Blinken am Laufwerk hört auf, und in den Variablen sind folgende Informationen gespeichert:

A: Nummer des Fehlers — C: Spur
B$: Fehlerbezeichnung im Klartext — D: Sektor

Jedem Fehler ist eine bestimmte Nummer zugeordnet. So hat die Meldung »File not found« die Nummer 62. »OK« und 0 sind Kennzeichen eines fehlerfreien Zugriffs auf die Diskette. Durch die Angabe der Spur- und Sektornummer erfahren wir außerdem, an welcher Stelle auf der Diskette der Fehler aufgetreten ist. Diese Angaben werden jedoch erst bei bestimmten Diskettenzugriffen interessant.

Beispiele für die Behandlung der häufiger auftretenden Fehler sind in unserem Programm »Floppyfehler« (Listing 1) dargestellt.

Das Programm arbeitet mit sequentiellen Dateien: Eingabe und Speicherung von Adressen auf Diskette. Zu Beginn des Programms muß ein Dateiname angegeben werden. Befindet sich unter diesem Namen bereits eine Datei auf der Diskette, so werden alle gespeicherten Adressen auf dem Bildschirm ausgegeben. Wenn die Datei noch nicht existiert, können Sie Adressen eingeben, die unter dem gewählten Namen auf Diskette gespeichert werden. Das Programm ist nicht sehr komfortabel. Es dient auch nur zur Demonstration der Fehlerbehandlung.

Im Teil A (Zeile 1000 bis 1530) wird eine Tabelle mit allen Fehlernummern angelegt, auf die das Programm reagieren soll. Die Variable AF enthält die Anzahl der Nummern. Außerdem muß an dieser Stelle der OPEN-Befehl für das Lesen der Fehlermeldung stehen.

Teil E (ab Zeile 9000) holt die Fehlermeldung und vergleicht sie mit den gespeicherten in der Tabelle. Die ersten vier Fehler werden gleich hier abgefangen. Dies sind Fehler, deren Behandlung unabhängig vom Programm gleich sind. So wird auf das Fehlen einer Diskette im Laufwerk meist mit der Aufforderung, eine solche einzulegen, reagiert. Ist die Fehlernummer in der Tabelle nicht enthalten, bricht das Programm ab.

Dateien werden wie Programme mit ihrem Namen auf der Diskette gespeichert. Durch die OPEN-Anweisung in Teil B teilen wir dem Computer unter anderem mit, welche Datei wir lesen möchten. Interessant sind in diesem Programm nach der automatischen Fehlerbehandlung nur zwei Meldungen.

1. Fehlernummer 0:
Die Datei ist vorhanden. Sie wird daraufhin gelesen und die Adressen auf dem Bildschirm ausgegeben.
2. Fehlernummer 62:
Die Datei ist nicht vorhanden. Wir können gleich mit der Eingabe der Adressen über die Tastatur beginnen.

Alle anderen Fehler sind schon im Teil E abgefangen worden. Zur Überprüfung, ob diese Fehler beseitigt wurde, wird die OPEN-Anweisung erneut ausgeführt.

Teil C (ab Zeile 2200) befaßt sich mit der Eingabe der Adressen. Die Gestaltung dieses Teils ist abhängig von dem zu lösenden Problem. Wir haben deshalb dort keine besondere Fehlerbehandlung vorgesehen.

Mit der OPEN-Anweisung in Teil D (ab Zeile 2400) teilen wir dem Computer mit, unter welchem Namen die Adressen gespeichert werden sollen. Wiederum sind zwei Meldungen interessant.

1. Fehlernummer 0:
Die Datei ist nicht vorhanden. Wir geben die Adressen deshalb gleich aus und legen damit die Datei an.
2. Fehlernummer 63:
Die Datei existiert bereits. In diesem Fall geben wir ihr einen anderen Namen. Der alte Name wird lediglich um den Zusatz ».BAK« erweitert. Tritt dann bei der Speicherung der Adressen ein schwerwiegender Fehler auf (Stromausfall), so haben Sie immer noch die alte Datei (mit dem Zusatz).

Teil A und Teil E können Sie fast unverändert in Ihre eigenen Programme übernehmen. Ändern Sie den Inhalt von AF entsprechend der von Ihnen gewählten Anzahl von Fehlernummern. Die automatisch in Teil E abzufangenden Fehler müssen die ersten in der Liste der DATA-Zeilen sein. Programmieren Sie für jeden dieser Fehler im Teil E ein entsprechendes Unterprogramm und ergänzen Sie die Reihe der Sprungadressen in Zeile 9100. (Peter Aurich/ef)

```
1000 REM *** TEIL A                                <001>
1001 :                                             <215>
1010 AF=6                                          <032>
1020 DIM EN(AF)                                    <203>
1100 DATA 26                                       <086>
1110 DATA 72                                       <072>
1120 DATA 74                                       <114>
1130 DATA 64                                       <116>
1140 DATA 62                                       <094>
1150 DATA 63                                       <120>
1500 FOR I=1 TO AF                                 <150>
1510 READ EN(I)                                    <007>
1520 NEXT I                                        <078>
1530 OPEN 14,8,15                                  <146>
1540 :                                             <248>
1600 REM *** TEIL B                                <099>
1601 :                                             <053>
1610 DIM N$(300),S$(300),O$(100),T$(100)           <092>
2000 AA=0                                          <030>
2100 PRINT CHR$(147)                               <097>
2110 INPUT "DATEINAME: ";DN$                       <028>
2120 PRINT                                         <190>
2130 OPEN 1,8,8,DN$+",S,R"                         <097>
2140 GOSUB 9000                                    <128>
2150 IF A=62 THEN 2300                             <215>
2160 IF A=0 THEN 2200                              <061>
2170 CLOSE 1                                       <149>
2180 GOTO 2100                                     <112>
2190 :                                             <134>
2200 REM *** TEIL 3                                <130>
2210 INPUT#1,AA                                    <135>
2220 FOR I=1 TO AA                                 <103>
2230 INPUT#1,N$(I),S$(I),O$(I),T$(I)               <134>
2240 PRINT N$(I) : PRINT S$(I)                     <165>
2250 PRINT O$(I) : PRINT T$(I)                     <187>
2260 PRINT                                         <074>
2270 NEXT I                                        <066>
2300 CLOSE 1                                       <023>
2310 AA=AA+1                                       <080>
2320 INPUT "NAME: ";N$(AA)                         <134>
2330 INPUT "STRASSE: ";S$(AA)                      <140>
2340 INPUT "ORT: ";O$(AA)                          <081>
2350 INPUT "TELEFON: ";T$(AA)                      <159>
2360 PRINT                                         <176>
2370 INPUT "WEITER MACHEN (J/N)";E$                <069>
2380 IF E$="J" THEN GOTO 2310                      <136>
2390 :                                             <080>
2400 REM *** TEIL D                                <145>
2401 :                                             <091>
2410 OPEN 1,8,8,DN$+",S,W"                         <133>
2420 GOSUB 9000                                    <154>
2430 IF A=0 THEN 2600                              <081>
2440 CLOSE 1                                       <165>
2450 IF A<>63 GOTO 2400                            <180>
2500 PRINT#14,"S:"+DN$+".BAK"                      <090>
2510 PRINT#14,"R:"+DN$+".BAK="+DN$                 <059>
2520 GOTO 2400                                     <246>
2600 PRINT#1,AA                                    <097>
2610 FOR I=1 TO AA                                 <241>
2620 PRINT#1,N$(I):PRINT#1,S$(I)                   <006>
2630 PRINT#1,O$(I):PRINT#1,T$(I)                   <113>
2640 NEXT I                                        <184>
2650 CLOSE 1 : CLOSE 14                            <150>
2660 PRINT "PROGRAMM BEENDET"                      <155>
3000 END                                           <208>
3010 :                                             <192>
9000 REM *** TEIL E                                <145>
9001 :                                             <087>
9010 INPUT#14,A,B$,C,D                             <223>
9020 IF A=0 THEN RETURN                            <027>
9030 PRINT                                         <242>
9040 I9=1                                          <213>
9050 IF EN(I9)=A THEN 9100                         <124>
9060 I9=I9+1                                       <192>
9070 IF I9<=AF THEN 9050                           <212>
9080 PRINT A;" ";B$;"{2SPACE}";"SPUR:";C;"
      SEKTOR:";D                                   <084>
9090 STOP : REM PROGRAMMABBRUCH                    <106>
9100 ON I9 GOTO 9200,9300,9400,9500                <081>
9110 RETURN                                        <022>
9120 :                                             <206>
9200 REM *** FEHLERMELDUNG 1                       <174>
9210 PRINT "DIE DISKETTE IST SCHREIBGESCHU
     ETZT" : PRINT                                 <185>
9220 PRINT "WENN SIE MIT DER DISKETTE ARBE
     ITEN WOLLEN"                                  <004>
9230 PRINT "{2SPACE}DANN ENTFERNEN SIE DEN
      SCHUTZ"                                      <072>
9240 GOSUB 9900                                    <004>
9250 RETURN                                        <164>
9260 :                                             <092>
9300 REM *** FEHLERMELDUNG 2                       <028>
9310 PRINT "DIE DISKETTE ODER DAS INHALTSV
     ERZEICHNIS SIND VOLL"                         <164>
9320 PRINT "{2SPACE}LEGEN SIE EINE ANDERE
     DISKETTE EIN"                                 <169>
9330 GOSUB 9900                                    <094>
9340 RETURN                                        <254>
9350 :                                             <182>
9400 REM *** FEHLERMELDUNG 3                       <136>
9410 PRINT "ES BEFINDET SICH KEINE FORMATT
     IERTE"                                        <010>
9420 PRINT "{2SPACE}DISKETTE IM LAUFWERK"          <071>
9430 GOSUB 9900                                    <194>
9440 RETURN                                        <098>
9450 :                                             <026>
9500 REM *** FEHLERMELDUNG 4                       <246>
9510 PRINT "DIESE DATEI EXISTIERT BEREITS
     ALS PROGRAMMDATEI"                            <114>
9520 PRINT "{2SPACE}AUF DER DISKETTE"              <037>
9530 PRINT "WAEHLEN SIE EINEN ANDEREN NAME
     N"                                            <044>
9540 GOSUB 9900                                    <050>
9550 RETURN                                        <210>
9560 :                                             <138>
9900 REM *** TASTE ABWARTEN                        <219>
9910 PRINT : PRINT "WENN FERTIG - TASTE DR
     UECKEN"                                       <228>
9920 GET E$ : IF E$="" THEN 9920                   <035>
9930 RETURN                                        <080>
9940 :                                             <008>
```

Listing 1. Abfangen von Floppyfehlern. Bitte mit dem Checksummer (Seite 159) eingeben.

So machen Sie Ihre Programme schneller

Basic-Programme können nach drei Gesichtspunkten optimiert werden: strukturiert und gut lesbar, schnell oder speicherplatzsparend. In diesem Beitrag zeigen wir Ihnen, wie Ihre Basic-Programme durch geschickte Programmierung schneller werden.

Es gibt einige Dinge, die man sehr wohl einzeln, aber nicht alle gleichzeitig haben kann. Wirtschaftswachstum, Inflation und niedrige Arbeitslosigkeit lassen sich eben nicht unter einen Hut bringen.

Beim Programmieren gibt es in einer höheren Sprache wie Basic ebenfalls so ein magisches Dreieck: strukturierte Programme, minimaler Speicherbedarf und kürzere Laufzeiten.

Gut lesbare und klar gegliederte Programme brauchen oft viel Speicherplatz und sind relativ langsam. Deswegen soll dieser Beitrag Sie zum Experimentieren anregen, Ihre Programme zu beschleunigen.

Basic ist nicht gleich Basic

Als erstes wollen wir einen einfachen aber typischen Programmablauf überlegen, den man in mehreren Varianten programmieren kann. Übrigens: Wegen der besseren Lesbarkeit schreiben Sie die nachfolgenden Basic-Zeilen besser mit Leerstellen zwischen den Befehlen.

Ich weiß, es geht auch kürzer, aber bei meinen Experimenten spielt Speicherplatz keine Rolle und außerdem habe ich noch einen Hintergedanken, den ich aber erst später erklären kann. Nicht zuletzt will ich dadurch erreichen, daß die Laufzeiten der Programme mit denen bei Ihnen übereinstimmen.

Schalten Sie bitte alle Programmierhilfen (wie beispielsweise Hyper-Basic, Simons Basic) und Disketten-Beschleuniger aus, denn sie können die Ablaufgeschwindigkeit der Programme verändern.

Der C 64 hat eine interne Uhr, deren Zeit abgefragt, ausgedruckt und somit zur Zeitmessung herangezogen werden kann. Im Befehlssatz der Commodore-Handbücher finden Sie dazu die beiden Funktionen TI und TI$. Mein Zeitmessungsprogramm besteht aus zwei Zeilen.

Die »Stoppuhr« wird gestartet mit:

```
10 TI$="000000"
```

Sie wird am Ende des Programmes gestoppt mit:

```
1000 PRINT TI/60 " SEKUNDEN" :END
```

Zwischen diesen beiden Zeilen plazieren wir die Prüflinge, das heißt die Programme, deren Laufzeit wir messen wollen. Zur Prüfung geben wir erst die Zeile 15 mit einer einfachen Verzögerungsschleife ein.

```
15 FOR T=1 TO 100:NEXT T
```

Dieses Programm hat eine Laufzeit von 0,13 Sekunden.

Löschen Sie bitte wieder die Zeile 15. Als Programm, das wir in mehreren Versionen vorstellen wollen, habe ich mir einen Ablauf ausgesucht, der auch optisch verfolgbar ist. Auf dem Bildschirm soll nämlich der Buchstabe A gleich 374mal nebeneinander gedruckt werden.

Die Zahl 374 hat nichts Magisches an sich. Es sind ganz einfach 17 Zeilen voller »A's« auf dem Bildschirm. 17 Zeilen lassen uns genug Platz, um die gestoppte Zeit darunter gut lesbar anzuzeigen.

In der **Version 1** des Programms verwenden wir POKE-Befehle, mit denen wir das A und die dazugehörige Farbe auf den Bildschirm bringen, das heißt in den Bildschirmspeicher (Video-RAM) und den Farbspeicher (Color-RAM). Beim C 64 sind es die Speicherzellen 1024 und 55296. (Sollte Ihnen diese Vorgehensweise nicht ganz klar sein, dann schauen Sie bitte in den Commodore-Handbüchern nach – es ist dort gut beschrieben.) Der Buchstabe A hat den Bildschirm-Codewert 1. Als Farbe wähle ich die »Normalfarbe«. Der Farbcode für Blau ist 14. Das Programm beginnt mit dem Löschen des Bildschirms (Zeile 20) und POKEt dann in Zeile 40 das erste A in den ersten Platz des Bildschirms:

```
20 PRINT CHR$ (147)
40 POKE 1024+Z:POKE 55296+Z,14
```

In Zeile 14 finden Sie zusätzlich eine Variable »Z«. Wie geplant, soll das A 374mal gePOKEt werden. Also müssen wir die Zahl der Speicherzelle laufend um 1 erhöhen, damit das A nicht stets an die gleiche Bildschirmstelle geschrieben wird. Dazu erfinden wir die Variable Z, die zur Speicherzelle addiert wird. Wir setzen Z am Anfang des Programms (in Zeile 30) auf Null und zählen es in Zeile 50 um 1 weiter:

```
30 Z=0
50 Z=Z+1
```

Als nächstes müssen wir prüfen, ob Z den Schlußwert 374 erreicht hat. Wenn nicht, dann soll der nächste POKE-Befehl ausgeführt werden, das heißt wir springen auf Zeile 40 zurück. Ist dies 374mal geschehen, springt der Computer zu Zeile 1000, in der die Zeit gestoppt und auf dem Bildschirm ausgegeben wird. Also:

```
60 IF Z=374 THEN 1000
70 GOTO 40
```

Ich schlage vor, daß Sie das kleine Programm noch einmal LISTen, damit Sie es komplett sehen können.

Die auf dem Bildschirm erscheinende Laufzeit wird nur davon beeinflußt, was zwischen den Zeilen 10 und 1000 steht. Sie können vor der Zeile 10 dem Programm hinzufügen, was Sie wollen.

Dreifacher Ersatz für »PRINZ AT«

Ein erster Probelauf mit RUN bringt das gewünschte Ergebnis, nur eines ist noch unschön: Der PRINT-Befehl in Zeile 1000 druckt uns die Zeit oben in die 2. Zeile, wo sie schlecht erkennbar ist. Wir könnten sie in einer anderen Farbe drucken, aber ich habe einen besseren Vorschlag.

Wir brauchen ein Rezept, um mit einem PRINT-Befehl an einem ganz bestimmten Platz auf dem Bildschirm drucken zu können. Einige Basic-Dialekte kennen den Befehl »PRINT AT«. Welche Möglichkeit bietet uns das Basic von Commodore?

1) PRINT "{Cursor Down}{Cursor Right}"
2) PRINT TAB(X)

3) PRINT SPC(X)
Um zum Beispiel an den 3ten Platz der 20ten Zeile die Zeit zu drucken, müßten wir 18mal das inverse »Q« für »Cursor-abwärts« und einmal »Cursor-rechts« eingeben.

Mit TAB(X) geht es besser. Wir haben nur ein Problem, daß nämlich der höchste zulässige Wert für X nur 255 ist (X nennt man das »Argument«). Wir müssen deshalb zwei TAB-Befehle einfügen, um einen Abstand von 400 Leerstellen zu erzeugen.

```
1000 PRINT TAB (255) TAB (155) TI/60"SEKUNDEN":END
```

Bei SPC(X) gilt dieselbe Begrenzung auf 255 für das Argument. Eine doppelte Verwendung von SPC geht natürlich auch, allerdings zählt SPC nicht vom Anfang der Zeile, sondern ab der letzten Cursor-Position. Durch Rechnen oder einfach durch Probieren finden wir die Gesamtzahl von 397, das ergibt:

```
1000 PRINT SPC (255) SPC (142) TI/60 "SEKUNDEN":
END
```

Es gibt noch eine dritte Methode, um PRINT AT zu simulieren.

In der Speicherzelle 214 kann die Zahl einer Zeile hineingePOKEt werden, auf die mit einem nachfolgenden PRINT der Cursor gesetzt wird. Das gleiche gilt für die Position in einer Zeile mit der Speicherzelle 211. Versuchen Sie es mit der direkten Eingabe:

```
POKE 214,8:PRINT:POKE 211,4:PRINT"A"
```

Das druckt den Buchstaben A in die 4te Spalte auf der 9ten Zeile. Für unseren Anwendungsfall in Zeile 1000 müssen wir die Zahl 18 in die Speicherzelle 214 POKEn, die Speicherzelle 211 können wir vernachlässigen.

```
1000 POKE 214,18:PRINT:PRINT TI/60"SEKUNDEN":END
```

Alle drei Methoden sind gleichwertig, sowohl in der Auswirkung als auch beim Speicherbedarf. Ich bleibe im folgenden bei der »214-Methode«.

Zurück zur Version 1 des Testprogramms. Nach RUN erhalten wir eine Laufzeit von 10,48 Sekunden.

So wird's schneller

Das Auffüllen des Bildschirms mit A's geht recht langsam. Schon beim Zuschauen wird man ungeduldig. Der Teil in diesem Programm, welcher die Laufzeit am nachhaltigsten beeinflußt, ist die 374fache Wiederholung des POKE-Befehls. Bei einem POKE-Befehl ist die Umwandlung der Zahlen aus dem ASCII-Code sehr zeitaufwendig. Hoppla, was heißt denn das schon wieder, werden Sie jetzt sicher fragen.

Jede Zahl, die hinter einem POKE-Befehl steht, wird anfangs in Form von vier einzelnen ASCII-Codezahlen gespeichert. Wenn das Programm abläuft, werden diese ASCII-Zahlen zuerst in ganze Zahlen, dann in Fließkomma-Zahlen umgewandelt (für den Fall, daß mit den Zahlen arithmetische Funktionen ausgeführt werden). Schließlich werden sie wieder in eine ganzzahlige POKE-Adresse umgewandelt - und das Ganze in unserem Fall 374mal.

Hier können wir die erste Verbesserung erreichen. Wenn wir eine so häufig vorkommende Zahl wie die POKE-Adresse in Zeile 40 am Anfang des Programms einer Variablen zuweisen, dann erfolgt die oben genannte Umwandlungssequenz nur einmal, nämlich am Anfang des Programms. Das Programm muß dann nur 374mal den Wert der Variablen im Speicher suchen, und das geht viel schneller. Wollen Sie es probieren? Ändern Sie für unsere **Version 2** die Zeilen 30 und 40.

Die Anfangsadresse im Bildschirmspeicher definieren wir als Variable mit dem zutreffenden Namen »VIDEO«, die des Farbspeichers mit »FARBE«.

In Zeile 30 erhalten wir die Werte:

```
30 Z=0:VIDEO=1024:FARBE=55296
```

Zeile 40 ergibt sich dann zwangsläufig:

```
40 POKE VIDEO+Z,1:POKE FARBE+Z,14
```

Alles andere bleibt gleich. Das Ergebnis nach RUN: eine Laufzeit von 7,3 Sekunden. Wir haben also eine Verkürzung der Zeit von zirka 3 Sekunden erzielt, das sind nicht weniger als 30 Prozent.

Eine Beschleunigung von 30 Prozent ist schon gut, aber noch nicht alles, was wir erreichen können. Da die Methode der vordefinierten Variablen so effektiv ist, wollen wir sie auf alle oft verwendeten Zahlen des Programms anwenden. Neben Z, VIDEO und FARBE gibt es noch die 1 für den Buchstaben A und die 14 für die Farbe sowie den Schlußwert 374 der Scheife. Sicher wissen Sie schon, wie es gemacht wird. Wir ändern folgende Zeilen:

```
30 Z=0: SCHLUSSWERT=374:VIDEO=1024:FARBE=55296:
BUCHSTA=1:DRUCK=14
40 POKE VIDEO+Z, BUCHSTA:POKE FARBE+Z,DRUCK
60 IF Z=SCHLUSSWERT THEN 1000
```

Das ist **Version 3** des Programms mit einer Laufzeit von 6,15 Sekunden. Wieder ergibt sich eine Verbesserung gegenüber der zweiten Version.

Bisher haben wir die Variablen wegen der guten Lesbarkeit mit langen und verständlichen Namen versehen. Das kostet natürlich Speicherplatz und Geschwindigkeit. Der Grund ist immer derselbe: Bei der Variablen VIDEO sind fünf Zeichen 374mal zu bearbeiten. Wenn wir die Variable nur »VI« nennen, sind erheblich weniger Zeichen zu bearbeiten.

In **Version 4** des Programms reduzieren wir alle langen Variablennamen auf zwei Zeichen.

```
30 Z=0: SC=374: VI=1024:FA=55296:BU=1:DR=14
40 POKE VI+Z,BU:POKE FA+Z,DR60 IF Z=SC THEN 1000
```

Ergebnis: 5,63 Sekunden. Gegenüber der Version 1 des Programms haben wir jetzt schon eine Verbesserung von 46 Prozent erreicht.

Treiben wir das Spiel in **Version 5** noch weiter und reduzierten alle zweistelligen Variablennamen auf einstellige. Damit müßte das Programm noch schneller werden:

```
30 S=374: V=1024: F=55296:B=1:D=16
40 POKE V+Z,B:POKE F+Z,D60 IF Z=S THEN 1000
```

Die erzielte Verbesserung ist zwar nicht überwältigend, aber meßbar. Der C64 braucht nun 5,51 Sekunden.

Nun können wir eine erste generelle Regel für schnellere Basic-Programme aufstellen.

Regel 1

* Häufig vorkommende Zahlen, Adressen oder Variablen werden am Anfang des Programms vordefiniert. Dies ist besonders wichtig, wenn sie innerhalb von Schleifen verwendet werden.
* Variablen, die sich im Lauf des Programms verändern, werden trotzdem vordefiniert. Achten Sie darauf, daß der gewählte Ausgangswert nichts Negatives anrichten kann.
* Die Zahl, die am häufigsten vorkommt, sollte als erste vordefiniert werden (damit sie schneller »gefunden« werden kann).
* Variablennamen sollen möglichst einstellig, höchstens zweistellig sein.

Halt, löschen Sie das Programm noch nicht! Wir liegen noch unter 50 Prozent mit unseren Verbesserungen. Was können wir am bisherigen Programm noch ändern?

Überlegen Sie einmal, wie man Buchstaben noch auf den Bildschirm bringen kann.

In **Version 6** des Programms ersetzen wir die POKE-Befehle für die Buchstaben durch eine PRINT-Anweisung, die zudem noch den Vorteil hat, daß keine Farbanweisung nötig ist. Den Buchstaben A wollen wir in dieser Version zu-

nächst einmal mit seiner ASCII-Codezahl 65 angeben. Ein Semikolon setzt alle A's hintereinander.

```
40 PRINT CHR$(65);
```

Für die Schleife brauchen wir nur die Variablen Z und S einzubinden:

```
30 Z=0:S=374
```

Alle anderen Zeilen bleiben unverändert.

Nach RUN sehen wir als Ergebnis: 4,05 Sekunden. Das ist eine weitere Verbesserung von 1,5 Sekunden. Unser Programmbeispiel wird durch die Verwendung von PRINT statt POKE stark beschleunigt. Das funktioniert deswegen besonders gut, weil alle Buchstaben automatisch hintereinander gesetzt werden. Wenn wir jedesmal den Platz mit angeben müßten, wohin gePRINTet werden soll, wäre der Vorteil rasch verspielt.

Die Anweisung PRINT CHR$(65) ist zwar gut lesbar, aber wir haben vorher gelernt, daß das Vordefinieren von Variablen schneller ist. In **Version 7** machen wir das auch mit der PRINT-Variablen.

```
30 Z=0: S=374: A$=CHR$(65)
40 PRINT A$
```

Das Resultat ist wieder beeindruckend: 3,10 Sekunden. Das sind jetzt bereits 69 Prozent Verbesserung gegenüber der 1. Version.

Sicher ist jedem die direkte PRINT-Anweisung mit Gänsefüßchen am geläufigsten. Sie benötigt auch weniger Speicherplatz und macht sogar ein Vordefinieren unnötig. Wir wollen prüfen, ob die PRINT-Anweisung mit Gänsefüßchen auch schneller ist.

Ändern Sie für **Version 8** die Zeilen 30 und 40:

```
30 Z=0:S=374
40 PRINT "A";
```

Erstaunlicherweise bringt das fast gar nichts. Das Ergebnis von 3,03 Sekunden ist nur 0,07 Sekunden schneller. Die Erlärung liegt darin, daß beide Darstellungen, CHR$(65) und "A", denselben ASCII-Code verwenden. Damit ist der einzige Unterschied zwischen den Versionen 7 und 8 die Gesamtanzahl der Zeichen im Programm. Fassen wir zusammen:

Regel 2

- In Schleifen mit aneinandergereihten Druckanweisungen ist PRINT viel schneller als POKE.
- Die PRINT-Variablen sollen entweder vordefiniert oder im Anführungszeichen-Modus eingesetzt werden.

In der PRINT-Schleife (Zeile 40) und nachfolgender IF-Abfrage (Zeile 60) gibt es noch zwei Feinheiten. Das C64-Basic erlaubt es, statt IF...THEN GOTO... nur IF...THEN... zu schreiben, und das haben wir bisher auch gemacht.

Man kann aber auch IF...GOTO... verwenden. Eigentlich ist nicht zu erwarten, daß zwischen den beiden Schreibarten ein Zeitunterschied besteht. Der Fall ist tatsächlich fast akademisch zu nennen, wie **Version 9** unter Beweis stellt:

```
60 IF Z=S GOTO 1000
```

Laufzeit des C64: 3,0 Sekunden, also nur um 0,03 Sekunden schneller.

Version 10 ändert auch nur ganz wenig an der Geschwindigkeit, spart aber eine ganze Zeile und damit Speicherplatz.

```
60 IF Z<>S GOTO 40
70 entfällt
```

Ergebnis: 2,98 Sekunden Laufzeit – nur 0,02 Sekunden schneller.

Regel 3

- Bei Schleifen mit Sprunganweisungen ist IF...GOTO schneller als IF...THEN
- Prüfung auf Ungleichheit (< >) bietet Vorteile, wenn sie eine Zeile erspart.

Für die Fortgeschrittenen unter Ihnen gebe ich noch eine weitere Anregung, die wir mit unserem Prüfprogramm nicht testen können.

Eine Prüfung mit IF..THEN hängt oft von mehr als einer Bedingung ab. Zum Beispiel kann sie so lauten:

```
100 IF (A=1 AND B=2 AND C=3) THEN 999
110 GOTO 50
```

Zeile 100 prüft jedesmal, ob alle drei Bedingungen erfüllt sind. Erst nach dem THEN wird entschieden, ob das Programm auf Zeile 999 springt oder auf 110 weiterläuft. Nehmen wir an, A ist im 20sten Durchlauf erfüllt, B im 50sten Durchlauf, C aber erst im 300sten Durchlauf. Das Programm muß also 300mal alle drei Bedingungen nachprüfen. Wenn wir die Zeile 100 so schreiben:

```
100 IF C=3 THEN IF B=2 THEN IF A=1 THEN 999
```

dann bricht das Programm 300mal die Prüfung nach dem C sofort ab und geht in 110 weiter. B und A werden erst dann zur Prüfung herangezogen, wenn C=3 ist. Es ist wohl einzusehen, daß diese Schreibweise schneller ist.

Regel 4

Bei IF...THEN-Prüfungen mit mehreren Bedingungen sollen diese Bedingungen in einzelnen IF...THEN-Prüfungen hintereinander gesetzt werden. Dabei wird jene Bedingung an die erste Stelle gesetzt, welche am wahrscheinlichsten nicht erfüllt wird.

Ich habe mit Absicht bisher die 374malige Wiederholung der Schleife mit der Zählvariablen Z hochgezählt und mit IF...THEN beziehungsweise IF...GOTO den Schlußwert abgefragt. Das gab mir die Gelegenheit, die Schnelligkeit dieser Methode ganz auszureizen. Wir haben auch die ursprüngliche Laufzeit von 10,48 Sekunden auf 2,98 Sekunden reduziert!

Wir wollen noch eine andere Möglichkeit austesten, nämlich die Schleife mit FOR...TO...NEXT zu programmieren.

In **Version 11** des immer noch gleichen Programms ändern wir dazu die Zeilen 30, 50 und 60:

```
30 FOR Z=1 TO 374
50 NEXT Z
60 entfällt
```

Starten Sie diese Version. Huiiih -jetzt geht's ab! Der C 64 braucht nur noch 1,08 Sekunden. Gegenüber der IF...THEN-Schleife der Version 10 ist diese Version ungefähr 2 Sekunden schneller, auf die Zeit der Version 1 bezogen sind es sogar 89 Prozent.

Das einzige, was wir in der FOR...NEXT-Schleife noch verbessern können, ist die vielgeübte Praxis des Weglassens der Schleifenvariablen Z nach dem NEXT-Befehl.

```
50 NEXT
```

Der Geschwindigkeitsgewinn dieser **Version 12** ist nicht sehr groß, nämlich nur etwa 0,1 Sekunden. Aber wir wollen diese Methode doch in die nächste Regel mit aufnehmen.

Regel 5

Schleifen sollen nicht mit IF..THEN, sondern mit FOR..TO..NEXT gebildet werden. Die Schleifenvariable nach NEXT soll dann weggelassen werden, wenn es nicht zu Verwechslungen mit anderen Schleifen führen kann.

Nun bin ich fast am Ende meines Beschleunigungslateins. Eines bleibt allerdings noch, nämlich die bisher hochgepriesene Lesbarkeit des Programms zu opfern. Ich hoffe, Sie haben bisher meinem Eingangshinweis Folge geleistet und alles schön mit Leerzeichen geschrieben. Das behalten wir zunächst noch bei, im Gegenteil, wir wollen die Lesbarkeit noch erhöhen und REM-Erläuterungen einfügen. Ich schlage vor, die **Version 13** so auszuschmücken:

```
10 TI$="000000":REM UHR AUF NULL
12 REM*****************************
13 REM*                           *
14 REM*TEST-ROGRAMM*              *
15 REM*                           *
16 REM*****************************
20 PRINT CHR$(147);:REM ALLES LOESCHEN
30 FOR Z=1 TO 374:REM 374 ZEICHEN
40 PRINT"A";
50 NEXT
999 REM ZEIT AUSDRUCKEN
1000 POKE 214,18:PRINT TI/60 "SEKUNDEN":END
```

Das Programm sieht zwar sehr schön aus, aber REM-Erläuterungen kosten Zeit. Es ist um 0,2 Sekunden langsamer geworden. Deshalb entfernen wir alle REM-Zeichen und erhalten wieder Version 12.

Machen wir nun einen Schritt in die andere Richtung und entfernen alle Leerstellen und Abstände. Mit dieser **Version 14** möchte ich Ihnen zeigen, daß dies einen Einfluß auf die Laufgeschwindigkeit hat.

```
10 TI$="000000"
20 PRINTCHR$(147);
30 FORZ=1TO374
40 PRINT"A";
50 NEXT
1000 POKE214,18:PRINT:PRINTTI/60"SEKUNDEN":END
```

Die neue Laufzeit: 0,98 Sekunden.

In **Version 15** treiben wir die Schrumpfung zum Extrem, indem wir das Programm mit einem Minimum an Zeilen schreiben, also möglichst viele Befehle in eine Zeile packen.

Faszinierende Einzeiler

Sie wissen, die maximale Zeilenlänge beträgt 80 Zeichen beim C 64. Unser Programm können wir sogar in einer einzigen Zeile unterbringen - fast unglaublich, aber es geht. Sie müssen allerdings alle Abkürzungsmöglichkeiten ausschöpfen, die das Commodore-Basic bietet. Im Anhang der Commodore-Handbücher finden Sie eine Liste aller Abkürzungen beim Eintippen: <?> für PRINT, <C> und <SHIFT H> für CHR$, <G> und <SHIFT O> für GOTO etc. Die meisten Befehle lassen sich durch zwei Zeichen abkürzen. Im nachfolgenden Ausdruck ist das natürlich nicht zu sehen, weil nach dem Befehl LIST nicht mehr die Abkürzungen, sondern die ausgeschreibenen Befehle zu sehen sind. So kommen auch mehr als 80 Zeichen in die eine Zeile des nachfolgenden Listings:

```
10 TI$="000000":PRINTCHR$(147);:FORZ=1TO374:PRINT"
A";:NEXT:POKE214,18:PRINT:PRINT TI/60"SEKUNDEN":END
```

Und siehe da, diese »Kurzform« des Programms ist auch die schnellste Version. Der C 64 braucht nur 0,93 Sekunden. Diese letzte Beschleunigung basiert darauf, daß das Betriebssystem des Computers nur einmal einen Zeilenanfang und ein Zeilenende suchen und erkennen muß, statt sechsmal wie in der Version 14.

Das Ausnützen der vollen Kapazität einer Zeile bringt also nicht nur den Vorteil eines kleineren Speicherbedarfs, sondern auch Zeitgewinn.

Regel 6

- Programme ohne REM-Erläuterungen und ohne Leerstellen zwischen den Zeichen laufen schneller.
- Zur Reduzierung der Zeilenzahl sollen möglichst viele Befehle in eine Zeile geschrieben werden.

»Einzeiler« können auch Spaß und zugleich eine Herausforderung sein. Man sollte annehmen, daß mit einer Zeile nicht viel anzufangen sei. Weit gefehlt!

Ich schreibe die nachstehenden Einzeiler lesbar, das heißt mit Leerstellen. Sie müssen die Programme aber wieder mit allen Abkürzungstricks schreiben, sonst geht es schief.

Von A. Boyd (Manchester) stammt ein Primzahlen-Erzeuger, der für die obere Grenze von 65000 viele Stunden braucht.

```
1 FOR N=1 TO 65000:F=0:FOR J=2 TO N-1:F=F+
((N-J*INT(N/J))=0):NEXT:X=-(F=0):
PRINTRIGHT$(STR$(X*N),6*X):NEXT
```

Ein anderer Einzeiler wurde von A.M. Simmelt (Sheffield) zur Konvertierung von Dezimal- in Dualzahlen geschrieben.

```
1 INPUT A:FOR I=14 TO 0 STEP-1:Z=A AND 2↑I:A=A-Z:
Z=Z/2↑I:Z$=RIGHT$(STR$(Z),1):PRINT Z$;:NEXT:GOTO1
```

Lassen wir es gut sein mit diesem Programmiersport und kehren wir zurück zu einer abschließenden Betrachtung der Zeitgewinne.

Wir haben in Version 1 mit 10,48 Sekunden begonnen. Diese Laufzeit wurde ohne Änderung des Programmresultats stetig verkürzt, bis wir schließlich in Version 15 bei 0,93 Sekunden gelandet sind. Ich nenne diese Beschleunigung um 90 Prozent schlicht und einfach spektakulär.

Mehr kann ich nicht herausholen, es sei denn man programmiert die Routine in Maschinensprache.

Es soll Ihnen hier kein Machinensprache-Kurs zugemutet werden. Doch ein Programm in Maschinensprache besteht ebenso aus Befehlen, Adressen und Variablen wie ein Basic-Programm, nur sind diese in einem speziellen Zahlencode geschrieben. Dieser Zahlencode muß in den Arbeitsspeicher geladen werden. Die für uns einfachste Möglichkeit besteht darin, die Zahlen in eine Speicherstelle hineinzuPOKEn. Damit wir aber nicht unmäßig viele POKE-Befehle schreiben müssen, legen wir alle Code-Zahlen hinter DATA-Befehle und holen sie dann mit READ in eine einzige POKE-Schleife. Ich betone das deswegen, weil dieses Einlesen natürlich nicht zu dem Testprogramm gehören darf, dessen Laufzeit wir messen wollen. Das Testprogramm selbst sitzt zwischen den drei Zeilen der »Stoppuhr«. Das heißt, genauer gesagt sitzt das Programm in den Speicherzellen, in die wir es hineinPOKEn. Aber erst nach dem Starten der Stoppuhr rufen wir es auf. Der dem RUN entsprechende Befehl für Maschinensprache heißt SYS.

Wie Sie gleich sehen werden, fängt unser Testprogramm ab Speicherzelle 7168 an:

```
10 TI$="000000"
20 PRINT CHR$ (147)
30 SYS 7168
1000 POKE 214,18:PRINT:PRINT TI/60"SEKUNDEN":END
```

Ab Zeile 2000 plazieren wir das Programm, welches uns das Maschinenprogramm einliest. Um mit dem Einlesen zu beginnen, setzen wir noch eine Umleitung vor das Meßprogramm:

```
5 GOTO 2000
```

In Zeile 2000 löschen wir den Bildschirm. Zeile 2010, 2020 und 2030 liest die Codezahlen, die in den Zeilen 2050 bis 2090 stehen, und POKEt sie in die Speicherplätze 7168

bis 7200. Sobald die Zahlen eingelesen sind, können Sie beispielsweise das Meßprogramm mit dem Befehl GOTO 10 (direkt eingeben) starten.

Im nachstehenden Programm wird das noch etwas eleganter gelöst. Zuerst meldet das Programm das Ende des Einlesens (Zeile 2100 und 2101). Dann kommt die Anweisung, wie das Meßprogramm zu starten ist, nämlich durch Drücken irgendeiner Taste (abgefragt durch eine GET-Schleife). Wenn eine Taste gedrückt wird, springt das Programm auf Zeile 10 (das geschieht in Zeile 2160)

```
   5 GOTO 2000
  10 TI$="000000"
  20 PRINT CHR$(147)
  30 SYS 7168
1000 POKE 214,18:PRINT:PRINT TI/60"SEKUNDEN":END
2000 PRINT CHR$(147)
2010 FOR A=7168 TO 7200
2020 READ B
2030 POKE A,B
2040 NEXT
2050 DATA 162,0,169,1,157
2060 DATA 0,4,169,14,157,0,216
2070 DATA 232,224,0,208,241,169,1,157
2080 DATA 254,4,169,14,157,254,216
2090 DATA 232,224,120,208,241,96
2100 PRINT"DAS MASCHINENPROGRAMM"
2110 PRINT"IST JETZT EINGELESEN.":PRINT
2120 PRINT"ZUM STARTEN DES PRO-"
2130 PRINT"GRAMMS" CHR$(18)"TASTE"CHR$(146)"
     DRÜCKEN"
2140 GET A$
2150 IF A$=""THEN 2140
2160 GOTO 10
```

Version	Programmier-Methode	Laufzeit (Sek.)
1	Buchstaben POKEn, Zählschleife mit IF ... THEN	10,48
2	POKE-Adressen vordefinieren	7,30
3	alle Variablen vordefinieren	6,15
4	Variablen mit 2 Buchstaben	5,63
5	Variablen mit 1 Buchstaben	5,51
6	PRINT CHR$ statt POKE	4,05
7	CHR$ vordefinieren	3,10
8	PRINT "A"	3,03
9	Schleife mit IF-GOTO statt IF-THEN	3,00
10	IF Z < > S statt IF Z = S	2,98
11	Schleife mit FOR-NEXT	1,08
12	NEXT ohne Zählvariable	1,00
13	Listing mit REMs	1,20
14	Listing ohne REM, ohne Abstände	0,98
15	Alles in eine Zeile	0,93
16	Maschinensprache	0,066

Tabelle 1. Ein Überblick zu den Laufzeiten der Programm-Versionen für »Buchstaben auf dem Bildschirm«

Starten Sie das Programm und Sie werden eine Überraschung erleben: die Laufzeit beträgt nur 0,066 Sekunden. Ich hoffe, daß ich Sie mit dem Virus der Maschinensprache zumindest ein wenig infiziert habe.

Eine Übersicht aller bisher genannten Versionen und die entsprechenden Laufzeiten finden Sie in Tabelle 1.

Wir wollen uns nun einem anderen Thema zuwenden, und im folgenden ein paar Rechenbeispiele (arithmetische Funktionen) untersuchen und beschleunigen. Als erste nehmen wir uns in **Version 17** die Multiplikation vor. Die Messung der Laufzeit erfolgt auf dieselbe Weise wie bei allen Programmen vorher. Deshalb bleiben die Zeilen 10, 20

Schneller Rechnen

und 1000 gleich. Die Multiplikation soll 300mal ausgeführt werden (Zeile 30). Dann wird das Ergebnis auf dem Bildschirm ausgegeben (Zeile 60).

```
30 FOR Z=1 TO 300
50 NEXT
60 PRINT A
```

Als Multiplikation nehmen wir den Extremfall, eine kurze Zahl multipliziert mit einer langen.

```
40 A=3*0.123456789
```

Nach RUN bleibt der Bildschirm zuerst leer, bis endlich nach 11,85 (14,15) Sekunden das Ergebnis der Multiplikation und die Laufzeit erscheinen.

In **Version 18** vertauschen wir die beiden Zahlen, die in Zeile 40 multipliziert werden.

```
40 A=0.123456789*3
```

Das führt natürlich nach Adam Riese zum gleichen Resultat wie vorher, aber die Laufzeit ist kürzer. Wir gewinnen 0,44 Sekunden. Dieser Gewinn ist nicht überwältigend, aber überraschend.

Erinnern wir uns, wie wir eine Multiplikation auf dem Papier durchführen. Da ist die Rechnung vom zweiten Fall auch einfacher als die vom ersten. Für den Computer ergibt sich das gleiche Problem.

In **Version 19** nutzen wir noch eine Eigenheit der Commodore-Computer aus, die auf ihre amerikanische Herkunft zurückzuführen ist. Es ist nämlich erlaubt, eine Null vor dem Dezimalpunkt wegzulassen. Obwohl das mit der Multiplikation direkt nichts zu tun hat, bietet uns doch eine gute Gelegenheit, die Einsparung durch das Weglassen der Null auch zeitlich zu messen. Zeile 40 sieht jetzt so aus:

```
40 A=.123456789*3
```

Das bringt zwar wieder nicht sehr viel, nur 0,20 Sekunden, aber »Kleinvieh macht auch Mist«.

Eine ähnliche Verbesserung, die wir hier nicht ausprobieren, wird erzielt durch das Ersetzen einer alleinstehenden Null durch einen Punkt, zum Beispiel:
statt IF X=0 THEN
jetzt IF X=. THEN

Regel 7

- Bei einer Multiplikation sollte die längere Zahl vor der kürzeren stehen (langer Multiplikant, kurzer Multiplikator).
- Eine einzelne Null wird durch einen Punkt ersetzt und die Null vor dem Dezimalpunkt wird weggelassen.

In **Version 20** ersetzen wir in Zeile 40 beide Zahlen durch Buchstaben, die wir in einer neuen Zeile 25 definieren.

```
25 B=.123456789:C=3
40 A=B*C
```

Dieser Ablauf endet nach 1,48 Sekunden, das heißt wir gewinnen 12,23 Sekunden. Also Regel 1 unbedingt beachten!

Eine andere beachtenswerte arithmetische Funktion ist das »Potenzieren« (Quadrat- oder Kubikzahlen), ausgelöst durch das Zeichen ↑. **Version 21** erzielen wir durch Löschen der Zeile 25 und Änderung der Zeile 40:

```
40 A=4↑3
```

»Vier hoch drei« ergibt 64 und benötigt 10,53 Sekunden. In **Version 22** wollen wir sehen, ob vordefinierte Variable ebenso einschlagen wie bei der Multiplikation.

```
25 B=4:C=3
40 A=B↑C
```

Diesmal sind wir nur 0,22 Sekunden schneller. Aber wir geben nicht auf! **Version 23** gleicht alles wieder aus, und zwar durch den simplen Trick, daß wir das Potenzieren in seine Grundelemente zerlegen.

Vier hoch drei (4↑3) ist dasselbe wie vier zweimal mit sich selbst multipliziert (4*4*4).

```
25 B=4
40 A=B*B*B
```

Das Programm braucht jetzt nur noch 1,68 Sekunden, ist also 8,31 Sekunden schneller geworden.

Regel 8

Die Funktion Potenzieren sollte durch Mehrfach-Multiplikation ersetzt werden.

In Tabelle 2 finden Sie eine Übersicht der Versionen 17 bis 23 mit den zugehörigen Laufzeiten.

Version	Programmier-Methode	Laufzeit (Sek.)
17	Multiplikation, lang x kurz	14,15
18	Multiplikation, kurz x lang	13,71
19	Null weglassen	13,51
20	Variablen vordefinieren	1.48
21	Potenzieren (4 hoch 3) mit ↑	10,53
22	Variable vordefinieren	10.31
23	4 x 4 x 4 statt 4 ↑3	2,01

Tabelle 2. Die Laufzeiten der Programm-Versionen für »arithmetische Funktionen«

Als letztes Objekt möchte ich oft aufgerufene Unterprogramme messen. Wir erreichen dies durch Ändern der letzten Version 23.

Version 24: Die Definition der Variablen (Zeile 25) und der Multiplikation (Zeile 40) verschieben wir als Unterprogramm ans Ende des Programms und springen innerhalb der 300fachen Schleife mit GOTO auf diese Zeilen.

Die Zeile 25 wird gelöscht.

```
30 FOR Z=1 TO 300
40 GOTO 40000
```

Alles andere bleibt gleich und neu hinzu kommt:

```
40000 B=4
50000 A=B*B*B
60000 GOTO 50
```

Es ist nicht weiter erstaunlich, daß dieser Umbau die Version 24 gegenüber Version 23 verlangsamt. Aber merken Sie sich die Laufzeit von 3,28 Sekunden. Als nächstes ersetzen wir die beiden GOTO-Zeilen durch GOSUB-RETURN.

```
40 GOSUB 40000
60000 RETURN
```

Diese **Version 25** spart 0,15 Sekunden. GOSUB ist schneller als GOTO! Sie haben vielleicht schon gelesen, daß oft gebrauchte Unterprogramme am Anfang eines Programms stehen sollen. Den Grund dafür will ich Ihnen mit den nächsten beiden Versionen vorführen.

Version 26 macht dies zunächst für die GOTO-Version. Wir bauen sie auf der Version 24 auf, mit folgenden Änderungen:

Die Zeitmessung lassen wir in den Zeilen 10, 20 und 1000, die Schleife und den Ausdruck des Resultats in den Zeilen 30, 50 und 60.

Nur beim Unterprogramm streichen wir alle Nullen der Zeilennummern, so daß es jetzt in den Zeilen 4, 5 und 6 steht. Um zu vermeiden, daß das Programm gleich mit dem Unterprogramm beginnt, fügen wir in Zeile 3 noch eine Umleitung ein, die sofort auf Zeile 10 springt.

Schließlich brauchen wir noch den Sprung in das Unterprogramm, den wir in Zeile 33 setzen. Das Ganze sieht folgendermaßen aus:

```
   3 GOTO 10
   4 B=4
   5 A=B*B*B
   6 GOTO 50
  10 TI$="000000"
  20 PRINT CHR$(147)
  30 FOR Z=1 TO 300
  33 GOTO 4
  50 NEXT
  60 PRINT A
1000 POKE 214,18:PRINT:PRINT TI/60 "SEKUNDEN":END
```

Nach RUN erhalten wir 3,1 Sekunden. Gegenüber Version 24, unserem Vergleichsobjekt, sparen wir 0,18 Sekunden.

Dasselbe passiert, wenn wir in **Version 27** die GOTOs durch GOSUB-RETURN ersetzen.

```
 6 RETURN
33 GOSUB 4
```

Gegenüber der anderen GOSUB-Version 25 sparen wir 0,17 Sekunden.

Regel 9

* Der Aufruf von Unterprogrammen ist mit GOSUB schneller als mit GOTO.
* Häufig gebrauchte Unterprogramme gehören an den Anfang des Programms. Sie müssen dann allerdings zuerst mit einem GOTO umgangen werden.

Einen Überblick über die Laufzeiten der »Unterprogramme« erhalten Sie in Tabelle 3.

Ich bin überzeugt, daß in Basic noch mehr spektakuläre Zeitgewinne stecken. Falls Sie eine Regel 10 oder noch mehrere entdecken, schreiben Sie doch an die Redaktion Sonderhefte.

(Dr. H. Hauck/U. Jürgens/kn)

Version	Programmier-Methode	Laufzeit (Sek.)
24	Unterprogramm am Ende, Sprung mit GOTO-GOTO	3,28
25	Unterprogramm am Ende, Sprung mit GOSUB-RETURN	3,13
26	Unterprogramm am Anfang, Sprung mit GOTO-GOTO	3,10
27	Unterprogramm am Anfang, Sprung mit GOSUB-RETURN	2,96

Tabelle 3. Die Laufzeiten der Programm-Versionen für »Unterprogramme«



Mein Computer versteht mich nicht!

Jeder Programmierer, ob Anfänger oder Profi, macht beim Programmeschreiben Fehler. Der Anfänger jedoch kennt meist weder den genauen Grund noch weiß er, wie er ihn beheben kann. Dem wollen wir mit einer ausführlichen Fehlertabelle und vielen Tips zur Suche der Fehlerursache abhelfen.

Obwohl dieser Artikel eigentlich mehr als Nachschlagewerk für den Notfall gedacht ist, schadet es nicht, wenn Sie ihn einmal komplett durchlesen. Sie haben so ein »Grundwissen«, auf das Sie bei Bedarf zurückgreifen können. Auch können Sie dann auf Zusammenhänge zwischen den einzelnen Fehlern schließen. Doch kommen wir nun zum Praxisteil:

Sie haben sich einen neuen C 64 besorgt, packen ihn voller Erwartung zu Hause aus, schließen ihn an, und los geht's mit dem Programmieren. Aber o weh! der Computer macht nicht das, was man von ihm verlangt. Es erscheinen Fehlermeldungen auf dem Bildschirm (erkennbar am vorangestellten »?« und daran, daß sich der Computer unerwartet mit »READY.« zurückmeldet). Was tun? Spätestens jetzt wird die Bedienungsanleitung zum C 64 zur Hand genommen, um dem Fehler auf die Schliche zu kommen. Aufmerksam liest man das entsprechende Kapitel, doch was versteht man? Nichts!! Lauter Fachausdrücke, von denen man noch nie etwas gehört hat, helfen auch nicht weiter. Genau an diesem Punkt soll dieser Artikel ansetzen, um Ihnen bei der Fehlersuche und deren Beseitigung unterstützend unter die Arme zu greifen. Jeder mögliche Fehler wird ausführlich behandelt. Dabei ist jedem Fehler ein Absatz gewidmet, der wie folgt aufgebaut ist:

- Fehlermeldung, also das, was auf dem Bildschirm erscheint und kurze Erklärung des Fehlers
- Klärung der Fachausdrücke
- Wie entsteht der Fehler, mit Beispielen?
- Wie geht man vor, um die Fehlerursache zu suchen?
- Wie wird der Fehler beseitigt?

Hinweis: Bevor Sie ein neues Beispiel eintippen, sollten Sie erst das alte mit »NEW« löschen.

Fehler im Betriebssystem:

Wenn Sie mit einer RS232-Schnittstelle arbeiten, so ist folgendes zu beachten: Nach einem `OPEN 1,2,...` also »Datenkanal zur RS232-Schnittstelle öffnen«, werden vom Computer sämtliche Variablen eines Basic-Programms gelöscht. Daher sollte dieser OPEN-Befehl unbedingt als erster Befehl in einem solchen Programm stehen.

?BAD SUBSCRIPT:

Der Fehler tritt auf, wenn versucht wird, eine Feldvariable anzusprechen, die nicht dimensioniert wurde.

Eine Feldvariable ist eine normale Variable wie »A« oder »TEST« mit zusätzlichem Index wie »A(5)« oder »TEST(X)«. Der Index in Klammern ist als Zeiger zu verstehen, der auf ein Element des Feldes mit dem Namen »A« oder »Test« zeigt. Felder, die mehr als elf Elemente enthalten, müssen mit dem Basic-Befehl DIM definiert werden. Felder, die weniger als elf Elemente enthalten, müssen nicht definiert werden. Zum Beispiel legt der Befehl DIM A(20) ein Feld mit insgesamt 21 Elementen fest (21 Elemente, weil die Feldvariable A(0) auch in der Dimensionierung enthalten ist). Wenn nach dieser Felddefinition versucht wird, der Feldvariablen »A(25)« einen Wert zuzuweisen: »A(25)=123«, erscheint die Fehlermeldung »BAD SUBSCRIPT« auf dem Bildschirm. Die gleiche Fehlermeldung erhält man, wenn der Feldvariablen »A(11)« ein Wert zugewiesen wird, ohne zuvor ein entsprechend großes Feld zu dimensionieren.

Tritt ein solcher Fehler in einem komplexeren Programm auf, ist zuerst zu überprüfen, wie groß der Index der Feldvariablen ist. Sollte er größer als 10 sein, ist die dazugehörige DIM-Anweisung zu suchen. Wenn eine solche nicht existiert, erweitern Sie eine der ersten Zeilen eben um diese DIM-Anweisung.

BREAK:

Bei dieser Meldung handelt es sich nicht um eine Fehlermeldung im herkömmlichen Sinn. Vielmehr macht der Computer darauf aufmerksam, daß das Programm oder ein Speicher- oder Ladevorgang absichtlich unterbrochen wurde. Diese Meldung existiert in zwei verschiedenen Versionen.

?BREAK ERROR zeigt an, daß beim Laden oder Speichern eines Programms die RUN/STOP-Taste gedrückt wurde.

?BREAK IN XXXX weist darauf hin, daß während eines Programmlaufes die RUN/STOP-Taste gedrückt wurde, oder daß sich in der angezeigten Zeile der Basic-Befehl »STOP« oder »BREAK« befindet. In einem solchen Fall kann das Programm mit dem Basic-Befehl »CONT« für »continue« fortgesetzt werden, vorausgesetzt es erscheint nicht die Fehlermeldung

?CAN'T CONTINUE:

In einem solchen Fall ist einer der folgenden Punkte eingetreten:

1) Das Programm wurde nicht, wie oben beschrieben, absichtlich unterbrochen, sondern der Computer selbst hat es angehalten, weil ein »SYNTAX ERROR« auftrat (der Basic-Übersetzer konnte einen Befehl im Programm nicht übersetzen/verstehen).

2) Im Direktmodus wurden mit dem Basic-Befehl »CLR« die Variablen gelöscht; das heißt Sie haben, nachdem das Programm unterbrochen wurde, folgendes eingegeben:

```
CLR <RETURN>
```

3) Am Programm wurde nach einer Unterbrechung etwas geändert.

4) Im Direktmodus trat nach einer Unterbrechung ein Fehler auf.

Beispiel: Sie starten ein beliebiges Basic-Programm, halten es mit der RUN/STOP-Taste an und geben ein

```
A <RETURN>
```

Der Computer meldet einen »SYNTAX ERROR«. Klar, den Befehl »A« gibt es nicht. Nun versuchen Sie das Programm mit »CONT« fortzusetzen. Der Computer gibt die Fehlermeldung »CAN'T CONTINUE« aus.

5) Wenn Sie den Befehl »CONT« eingeben, bevor ein Basic-Programm gestartet wurde, reagiert der Computer ebenfalls mit »CAN'T CONTINUE«.

?DEVICE NOT PRESENT:

Es wurde versucht, ein peripheres Gerät wie Drucker, Diskettenlaufwerk oder ähnliches anzusprechen, das entweder nicht eingeschaltet oder überhaupt nicht vorhanden ist oder nicht reagiert. Dieser Fehler tritt gewöhnlich nicht beim eigentlichen »OPEN«-Befehl auf, wie das folgende Beispiel zeigt, sondern dann, wenn versucht wird, das Gerät mit den Befehlen »GET#, INPUT#, PRINT#« anzusprechen. Beispiel:

```
OPEN 7,7 <RETURN>
PRINT#7,"ABCDEF" <RETURN>
```

Meldet der Computer einen »DEVICE NOT PRESENT«-Error, sollten Sie zunächst überprüfen, ob alle erforderlichen Geräte (Drucker, Diskettenlaufwerk und so weiter) eingeschaltet und mit dem Computer verbunden sind. Ist das der Fall, haben Sie mit Sicherheit eine falsche Geräteadresse (zweite Zahl hinter dem OPEN-Befehl) gewählt. Um den für die Fehlermeldung verantwortlichen OPEN-Befehl zu finden, lassen Sie sich die in der Fehlermeldung angegebene Zeile LISTen. Suchen Sie in dieser Zeile einen Basic-Befehl, dem unmittelbar das Nummernzeichen (#) und eine Zahl (Filenummer) folgt. Sind mehrere solcher Befehle vorhanden, ist die Zeile aufzuteilen und zwar so, daß jede Zeile nur einen Befehl mit Nummernzeichen enthält. Beispiel:

```
10 PRINT#4,CHR$(32):GET#8,A$
```

Diese Zeile ist in die folgenden zwei Zeilen aufzuteilen:

```
10 PRINT#4,CHR$(32)
11 GET#8,A$
```

Starten Sie nun das Programm erneut mit RUN und lassen sich wieder die in der Fehlermeldung angegebene Zeile LISTen. Im nächsten Schritt ist der OPEN-Befehl im Programm zu suchen, der die gleiche Filenummer (erste Zahl hinter dem OPEN-Befehl) hat, wie der Befehl (Zahl hinter dem Nummernzeichen), bei dem der Fehler aufgetreten ist. Die zweite Zahl hinter dem fehlerträchtigen OPEN-Befehl ist die falsche Geräteadresse. Damit Sie dort die richtige Geräteadresse einsetzen können, folgt eine Tabelle aller möglichen Geräteadressen:

1 Datasette
2 RS232
3 Bildschirm
4 Drucker
5 Drucker, falls im Drucker die Geräteadresse auf 5 umgestellt wurde
6 Plotter 1520
7 frei für externe Geräte
8 bis 11 Diskettenlaufwerke
12 bis 15 frei für externe Geräte

?DIVISION BY ZERO:

Dieser Fehlermeldung erscheint nur dann, wenn durch Null dividiert wurde. Aber Vorsicht! Es ist nicht immer auf Anhieb ersichtlich, daß der Nenner tatsächlich Null ist. Ist zum Beispiel bei der Funktion »A=B/(C*D*E*F)« eine der Variablen »C, D, E, F« Null, so ist natürlich der gesamte Nenner Null. Dieses Beispiel ist vielleicht trivial, aber was machen Sie, wenn bei der Funktion »A=TAN (PI/2)« ein »DIVISION BY ZERO«-Error erscheint? Um das zu verstehen, muß man wissen, wie der Computer die TAN-Funktion behandelt. Er berechnet nämlich nicht direkt den Tangens, sondern ermittelt ihn mit Hilfe der SIN- und COS-Funktion (tan(x) = sin(x)/cos(x)). Wenn für X PI/2 eingesetzt wird, ist der Nenner »COS (PI/2)« Null, und es erscheint die Fehlermeldung.

Um den »DIVISION BY ZERO«-Error zu beheben, ist jede Variable im Nenner, falls vorhanden, mit dem Basic-Befehl »PRINT«, gefolgt von der Variablen, zu überprüfen. Ist die Variable gefunden, die den Fehler verursacht hat, müssen Sie dafür sorgen, daß sie niemals den Wert Null annehmen kann. Dies läßt sich durch eine Abfrage vor der Funktion realisieren. Beispiel:

```
IF C*D*E*F <>0 THEN A=B/(C*D*E*F)
```

?EXTRA IGNORED:

Zeigt an, daß hinter dem Basic-Befehl INPUT weniger Variablen stehen, als eingegeben wurden. Beispiel:

```
10 INPUT A$,B$ <RETURN>
```

Nach der Eingabe wird dieser Einzeiler mit RUN <RETURN> gestartet. Es erscheint ein Fragezeichen (?) auf dem Bildschirm. Geben Sie nun ein:

```
TEST1,TEST2,TEST3 <RETURN>
```

Der Computer meldet einen »EXTRA IGNORED«-Error, denn »TEST1« wird in die Variable »A$« und »TEST2« in die Variable »B$« eingelesen. Für »TEST3« existiert keine Variable, und genau dies meldet der Computer. Häufig entsteht

eine solche Fehlermeldung unbeabsichtigt bei Zeichenketteneingaben, die Kommata enthalten. Man sollte daher darauf achten, wenn mit dem INPUT-Befehl gearbeitet wird, daß eine Zeichenkette, die in eine Variable eingelesen werden soll, kein Komma enthält. Durch das Komma wird nämlich dem Computer mitgeteilt, daß alle folgenden Zeichen bis zum nächsten Komma in die nächste Variable eingelesen werden sollen. Eine Zeichenkette darf neben dem Komma auch keine Doppelpunkte enthalten; denn alle Zeichen, die hinter dem ersten Doppelpunkt stehen, werden vom Computer verschluckt. Der gleiche Fehler tritt auch bei dem Befehl »INPUT #« auf, jedoch wird in diesem Fall keine Fehlermeldung ausgegeben. Übrigens noch ein Tip: Bei älteren C 64 darf der Cursor nicht nach oben oder unten verschoben werden, wenn der INPUT-Befehl eine Eingabe verlangt.

FILE DATA ERROR:

Diese Fehlermeldung erscheint, wenn im Programm versucht wird, Zeichenketten vom Diskettenlaufwerk, von der Datasette oder der Tastatur in eine numerische Variable einzulesen. Beispiel:

```
10 OPEN1,0:REM TASTATUR ALS EINGABEGERÄT ÖFFNEN
20 INPUT#1,A:REM AUF EINGABE WARTEN
30 CLOSE1:REM EINGABEGERÄT »TASTATUR« SCHLIESSEN
```

Zeile 10 öffnet einen Kanal mit der Filenummer 1 und setzt die Tastatur (Geräteadresse 0) als Eingabegerät.

In Zeile 20 wird auf die Eingabe gewartet. Es können beliebige Zahlen (keine Buchstaben) eingegeben werden. Nach dem Drücken der RETURN-Taste wird in Zeile 30 der Kanal mit der Filenummer 1 geschlossen.

Sind die drei oben stehenden Zeilen abgetippt, läßt sich das Programm mit RUN starten. Im Gegensatz zum normalen INPUT-Befehl ohne dem Nummernzeichen wird kein Fragezeichen ausgegeben. Geben Sie nun ein:

```
123 <RETURN>
456789 <RETURN>
```

und so weiter

Es erscheint, wie erwartet, keine Fehlermeldung. Nun versuchen Sie es mal mit »1A3«. Der Computer meldet einen »FILE DATA«-Error. Die erste Zahl »1« kann der numerischen Variablen »A« zugewiesen werden. Das zweite Zeichen jedoch ist keine Zahl, sondern ein alphanumerisches Zeichen beziehungsweise ein Buchstabe und läßt sich nicht einer numerischen Variablen zuordnen.

Der angezeigte Fehler kann sehr schnell gefunden werden, da die Zeile, in der der Fehler auftrat, mit der Fehlermeldung zusammen ausgegeben wird.

?FILE NOT FOUND:

Es wurde versucht, ein Programm- oder Daten-File zu laden, dessen Name nicht auf der momentan eingelegten Diskette existiert. Um diese Fehlermeldung kennenzulernen, tippen Sie einfach ein »LOAD"ALR",8«. Natürlich muß dazu ein Diskettenlaufwerk vorhanden, eingeschaltet und mit dem Computer verbunden sein. Erscheint eine solche Fehlermeldung innerhalb eines Programms, wird zusammen mit der Fehlermeldung die Zeile ausgegeben, in der der Fehler auftrat. Sollte in der fehlerhaften Zeile statt eines Namens hinter dem LOAD-Befehl nur eine alphanumerische Variable stehen, zum Beispiel »...:LOAD A$,8:...«, dann läßt sich dieser Name mit »PRINT A$« auf den Bildschirm bringen. Wenn sich aus irgendeinem Grund der Name nicht mehr feststellen läßt, kann er mit dem Befehl »SYS63123« noch einmal angezeigt werden. Übrigens: Die Meldung »FILE NOT FOUND« erscheint nur, wenn mit dem Diskettenlaufwerk gearbeitet wird. Bei Ladeversuchen mit der Datasette wird entweder nichts oder »DEVICE NOT PRESENT« ausgegeben.

?FILE NOT OPEN:

Es wurde versucht, mit den Befehlen »PRINT #, GET #, INPUT #« Zeichen an ein externes Gerät zu übermitteln oder von einem solchen zu lesen, ohne zuvor das File mit dem OPEN-Befehl geöffnet zu haben. Dabei ist es durchaus denkbar, daß eine falsche Filenummer (erste Zahl hinter dem OPEN-Befehl) hinter einem der oben stehenden Befehle oder hinter dem OPEN-Befehl eingesetzt wurde. Beispiel:

```
10 OPEN 5,4: REM DRUCKERKANAL ÖFFNEN
20 PRINT#4,"ABC":REM UND DIE ZEICHEN »ABC« DRUCKEN
30 CLOSE 4: REM DRUCKERKANAL SCHLIESSEN
```

Wird dieses Programm eingegeben und gestartet, so erhält man einen »FILE NOT OPEN«-Error, denn die Zeile öffnet einen Kanal mit der logischen Filenummer »5«. Die Zeilen 20 und 30 beziehen sich aber auf die logische Filenummer »4«.

Bei längeren ineinander verschachtelten Programmschleifen ist es häufig sehr schwierig, einen solchen Fehler zu beseitigen. Daher geht man einen anderen Weg. Man öffnet direkt vor dem PRINT #-Befehl in der gleichen Zeile einen Kanal mit der entsprechenden Filenummer (im Beispiel wird die Zeile 20 so ergänzt: 20 OPEN4,4:PR...). Jetzt muß der Kanal natürlich innerhalb des Programms wieder geschlossen werden. Wenn Sie unsicher sind, daß das tatsächlich gemacht wird, ist unmittelbar hinter dem PRINT #-Befehl der Kanal zu schließen (im Beispiel muß wieder die Zeile 20 ergänzt werden: 20 ..:..."ABC":CLOSE 4). Tritt der gleiche Fehler nach dem Start des Programms in einer anderen Zeile auf, in der ebenfalls die gleiche Filenummer benutzt wurde, so muß der zuletzt eingefügte CLOSE-Befehl gelöscht und hinter dem Befehl eingesetzt werden, der den neuen Fehler verursacht hat. So können Sie sich durch das gesamte Programm »hangeln«, bis es fehlerfrei arbeitet.

?FILE OPEN:

Diese Fehlermeldung ist genau das Gegenstück zu der Fehlermeldung »FILE NOT OPEN«. Erscheint der »FILE OPEN«-Error auf dem Bildschirm, wurde ein Kanal mit der gleichen Filenummer zweimal geöffnet. Das heißt es fehlt ein CLOSE-Befehl, der den zuerst geöffneten Kanal wieder schließt. Beispiel:

```
10 OPEN 4,4
20 OPEN 4,4
```

Der einfachste Weg, diesen Fehler zu beseitigen, ist es, vor dem OPEN-Befehl, in der Zeile, die hinter der Fehlermeldung angegeben ist, den entsprechenden Kanal mit dem CLOSE-Befehl zu schließen (im Beispiel: 20 CLOSE 4:OPEN 4,4). Sie werden sich jetzt zu Recht fragen, warum wird nicht einfach die Zeile 20 gelöscht? Die Antwort darauf ist ganz einfach. Hinter dem ersten OPEN-Befehl mit der Filenummer 4 könnte ja auch eine andere Geräteadresse stehen, durch die ein anderes Gerät angesprochen wird (zum Beispiel OPEN 4, 8 für Diskettenlaufwerk). Würde jetzt der zweite OPEN-Befehl (im Beispiel Zeile 20) gelöscht werden, so würde alles, was zum Drucker (Geräteadresse 4) gesendet werden soll, zum Diskettenlaufwerk geschickt. Daher seien Sie äußerst vorsichtig mit dem Löschen von Befehlen innerhalb eines Programms.

?FORMULA TOO COMPLEX:

Es ist fast unmöglich, diesen Fehler in ein Basic-Programm einzubauen. Trotzdem existiert die Fehlermeldung. Durch sie meldet der Computer, daß in einer Zeichenkettenaddition zu viele Klammerausdrücke stehen. Beispiel:

```
PRINT "A"+("B"+"B1"+("C"+"D"))
```

Die Anzahl der Zeichen innerhalb der einzelnen Zeichenketten ist völlig uninteressant. Wichtig ist, daß zweimal die

Zeichenkombination »+(« vorkommt. Denn der Computer versucht sich alles vor den jeweiligen Klammerausdrücken zu merken. Im Beispiel merkt sich der C64 das Zeichen »A«, berechnet »"B"+"B1"="BB1"« und versucht sich auch diese Zeichen zu merken, um im nächsten Schritt »"C"+"D"« auszurechnen. Soweit kommt er aber nicht, weil er sich maximal eine Zeichenkette merken kann (im Beispiel »A«) und gibt daher die Fehlermeldung »FORMULA TOO COMPLEX« aus.

Anmerkung: Verschachtelungen innerhalb einer Zeichenkettenaddition sind völlig überflüssig. Sie erhalten nur dann einen Sinn, wenn nicht nur addiert, sondern auch multipliziert oder dividiert wird, und genau dies ist mit Zeichenketten nicht möglich.

Die oben stehende Zeile könnte daher auch so aussehen:

```
PRINT "A"+"B"+"B1"+"C"+"D"
```

oder einfach

```
PRINT "A";"B";"B1";"C";"D"
```

oder noch einfacher

```
PRINT "A" "B" "B1" "C" "D"
```

Das Ergebnis ist immer das gleiche. Deshalb ist bei Zeichenkettenoperationen das Pluszeichen und vor allen Dingen das Arbeiten mit Klammern zu vermeiden.

Sollte trotzdem mal ein »FORMULA TOO COMPLEX«-Error erscheinen, ist ein fataler Fehler gemacht worden, der sich nur beseitigen läßt, indem man den Computer kurzzeitig ausschaltet.

?ILLEGAL DEVICE NUMBER:

Diese Fehlermeldung bedeutet, daß ein Befehl zu einem unzulässigen Gerät geschickt wurde. Mit anderen Worten: Die im Befehl angegebene Geräteadresse ist falsch. Zum Beispiel erzeugt »LOAD"TEST",0« einen »ILLEGAL DEVICE NUMBER«-Error, denn es ist unmöglich, ein Programm von dem Gerät mit der Geräteadresse »0« (Tastatur) zu laden.

Erscheint diese Fehlermeldung auf dem Bildschirm, ist die Beseitigung des Fehlers recht einfach. Lassen Sie sich die Zeile, die hinter der Fehlermeldung angegeben ist, LISTen und schauen sich alle LOAD- und SAVE-Befehle an. Ist eine der Geräteadressen hinter diesem Befehl »0, 2 oder 3«, dann haben Sie den Fehler gefunden. Sollte nicht bekannt sein, welche Gerätenummer welchem Gerät zugeordnet ist, dann schauen Sie in der Gerätenummern-Tabelle nach, die bei der Beschreibung der Fehlermeldung »DEVICE NOT PRESENT« aufgeführt ist.

ILLEGAL DIRECT:

Wenn versucht wird, den Basic-Befehl »GET« oder »INPUT« im Direktmodus einzugeben, dann meldet sich der Computer mit dieser Fehlermeldung. Der Direktmodus unterscheidet sich vom Programmodus dadurch, daß im Direktmodus jeder eingegebene Befehl ausgeführt wird, sobald man die RETURN-Taste drückt. Den Programmodus erkennt der C64 an einer Zahl (Zeilennummer) am Anfang einer Zeile. Wird hinter einer Zeile, die mit einer Zeilennummer beginnt, die RETURN-Taste gedrückt, dann wird die Zeile nicht ausgeführt, sondern in den Speicher des C64 geschrieben. Beispiel:

```
INPUT A$ <RETURN>
```

führt zur Fehlermeldung »ILLEGAL DIRECT«

Wird vor den INPUT-Befehl eine Zahl geschrieben »10 INPUT A$ < RETURN>«, dann akzeptiert der Computer diese Zeile und speichert sie. Wenn der INPUT-Befehl jetzt ausgeführt werden soll, ist RUN < RETURN> im Direktmodus einzugeben.

Da der »ILLEGAL DIRECT«-Error niemals in einem Programm auftauchen kann, erübrigt sich die Behebung dieses Fehlers.

ILLEGAL QUANTITY:

Diese Fehlermeldung zeigt an, daß die Berechnung einer Funktion eine Zahl ergab, die der Computer in dieser Form nicht verarbeiten kann. Arbeitet man mit Integer-Variablen (zum Beispiel A%), ist darauf zu achten, daß den Variablen niemals Zahlen zugewiesen werden, deren Wert 32000 überschreitet. Wird versucht, mit dem Basic-Befehl POKE eine Zahl zu speichern, deren Wert größer ist als 255 oder negativ, dann erscheint auch ein »ILLEGAL QUANTITY«-Error. Die gleiche Fehlermeldung gibt der C64 auch dann aus, wenn beim OPEN-Befehl eine logische Filenummer (erste Zahl hinter dem OPEN-Befehl) größer als 255 gewählt wurde, oder wenn aus einer negativen Zahl die Quadratwurzel gezogen werden soll. Alle folgenden Beispiele erzeugen einen »ILLEGAL QUANTITY«-Error:

```
A%=33000:REM DIE INTEGERVARIABLE A% IST GRÖSSER
32000
POKE 2,300:REM ES SOLL EINE ZAHL GESPEICHERT
WERDEN, DIE GRÖSSER IST ALS 255
POKE 2,-3:REM NEGATIVE WERTE LASSEN SICH NICHT
SPEICHERN
OPEN 300,4:REM FILENUMMER GRÖSSER 255
PRINT SQR(-1):REM AUS NEGATIVEN ZAHLEN KANN NICHT
DIE QUADRATWURZEL GEZOGEN WERDEN
```

Einen solchen Fehler im Programm aufzuspüren ist nicht einfach, denn in den meisten Fällen enthalten Variable die fehlerträchtigen Zahlen. Um einen solchen Fehler zu beseitigen, lassen Sie sich zunächst die Zeile LISTen, die den Fehler enthält. Danach ist jede in dieser Zeile vorkommende Variable mit dem Befehl PRINT auf den Bildschirm zu schreiben. Wenn bei diesem Versuch die Fehlermeldung angezeigt wird, ist zumindest die fehlerhafte Variable gefunden. Jetzt muß gegebenenfalls Schritt für Schritt der Programmablauf verfolgt werden, um die Stelle zu finden, an der der Variablen die falsche Zahl zugewiesen wurde.

I/O ERROR 1 bis 29:

Diese Fehlermeldung existiert nach dem Einschalten des Computers nicht. Nur dann, wenn in die Speicherzelle 157 die Zahl 64 oder 192 geschrieben wird (zum Beispiel POKE 157,64), gibt der Computer im Falle eines Fehlers die Fehlermeldung »I/O ERROR« aus. Dabei entspricht eine Zahl zwischen 1 und 29 hinter der Fehlermeldung den normal angezeigten Fehlern wie zum Beispiel »ILLEGAL QUANTITY«. Beispiel:

```
POKE157,192:LOAD"ABC",9
```

Existiert kein Diskettenlaufwerk mit der Gerätenummer 9, so führt das Beispiel zu der Fehlermeldung:

```
SEARCHING FOR ABC
I/O ERROR #5
?DEVICE NOT PRESENT
```

?LOAD ERROR:

Ein Programm oder Daten-File konnte nicht fehlerfrei von der Datasette oder von dem Diskettenlaufwerk geladen werden. Meldet der Computer diesen Fehler, wenn ein Programm abgearbeitet wird, so liegt der Fehler nicht am Programm, sondern an dem Gerät, von dem es geladen wurde. Versuchen Sie daher das gleiche Programm noch einmal zu laden. Scheitert auch dieser Versuch, so ist das Programm verloren. Meistens kommt ein »LOAD ERROR« dann vor, wenn entweder die Diskette oder das Band der Kassette mechanisch beschädigt ist.

?MISSING FILE NAME:

Weist darauf hin, daß beim LOAD- oder SAVE-Befehl kein Name angegeben worden ist, denn die Angabe eines Namens ist bei Operationen mit dem Diskettenlaufwerk unbe-

dingt erforderlich. »LOAD "",8 < RETURN>« führt folglich zur Fehlermeldung »MISSING FILE NAME«.

Tritt der Fehler innerhalb eines Programms auf, lassen Sie sich die Zeile, die hinter der Fehlermeldung angegeben ist, LISTen. Folgt dem entsprechenden LOAD- oder SAVE-Befehl ein Leerzeichen »" "«, dann ist zwischen den Gänsefüßen ein beliebiger Name einzusetzen. Schwieriger wird es, wenn der Befehl so aussieht:

```
...:load a$,8:...
```

Schauen Sie sich in einem solchen Fall den Inhalt der Stringvariablen (Zeichenkettenvariable) mit dem Befehl

```
PRINT A$ <RETURN>
```

an. Sie wird kein einziges Zeichen enthalten. Folglich muß unmittelbar vor dem entsprechenden Befehl der Stringvariablen A$ ein Name zugewiesen werden. Die Zeile (Beispiel) müßte dann so aussehen:

```
...:a$="zeichen":load a$,8:...
```

Häufig ist aber erwünscht, daß sich mit der gleichen Zeile unterschiedliche Programme laden lassen. In einem solchen Fall muß, bevor der LOAD- oder SAVE-Befehl ausgeführt wird, abgefragt werden, ob die Stringvariable A$ eine Zeichenkette enthält. Dazu muß das Programm um eine Zeile ergänzt werden, die die erforderliche »IF«-Abfrage enthält. Enthält A$ nichts, also nur einen Leerstring, so ist der Stringvariablen eine beliebige Zeichenkette zuzuweisen.

```
19 IF A$ = "" THEN A$ = "ZEICHEN"
20 ...:LOAD A$,8
```

NEXT WITHOUT FOR:

Erscheint, wenn zur Anweisung NEXT die entsprechende FOR-Anweisung fehlt. Dies kann verschiedene Gründe haben:

1) Im Programm befinden sich mehr NEXT- als FOR-Anweisungen.

2) Es wird versehentlich mit dem Basic-Befehl GOTO in eine FOR..NEXT-Schleife gesprungen.

3) Der Name der Schleifenvariablen (Variable hinter dem FOR-Befehl) in der FOR-Anweisung stimmt nicht mit derjenigen hinter der NEXT-Anweisung überein.

4) Es wurde innerhalb einer Schleifenkonstruktion mit dem Befehl GOSUB ein Unterprogramm aufgerufen, das eine NEXT- ohne entsprechende FOR-Anweisung enthält.

Fehler dieser Art lassen sich nur sehr schwer beheben. Vor allen Dingen dann, wenn das Programm lang und unübersichtlich geschrieben wurde. Der gesamte Programmablauf muß zu »Fuß« im Listing nachvollzogen werden. Zu jeder FOR-Anweisung ist die entsprechende NEXT-Anweisung zu suchen. Es muß überprüft werden, ob von irgendeiner Stelle in die Schleife gesprungen wird, in der der Fehler ausgegeben wurde.

Daher ist es sinnvoll, sich zu jedem Programm einen Ablaufplan (Flußdiagramm) anzufertigen, damit der Überblick nicht verlorengeht und Fehler, die nur schwer zu beheben sind, vermieden werden.

?NOT INPUT FILE:

Erscheint, wenn versucht wird, mit den Befehlen INPUT# und GET# aus einer Datei etwas zu lesen, die zum Schreiben geöffnet wurde, oder wenn als Filenummer (erste Zahl hinter dem OPEN-Befehl) »0« eingesetzt wird. Versuchen Sie mal im Direktmodus

```
OPEN0,3 <RETURN>
```

einzugeben. Der Computer meldet sofort einen »NOT INPUT FILE«-Error. Wenn eine Datasette zur Verfügung steht, erzeugt das folgende Beispiel die gleiche Fehlermeldung:

```
10 OPEN1,1,1,"TEST":REM BANDDATEI ZUM
SCHREIBEN ÖFFNEN
20 PRINT#1,"WORT":REM DAS WORT »WORT« IN DIE
DATEI SCHREIBEN
30 INPUT#1,A$:REM ERZEUGT DIE FEHLERMELDUNG
40 CLOSE1:REM KANAL SCHLIESSEN
```

Erscheint ein »NOT INPUT FILE«-Error, ist die hinter der Fehlermeldung angegebene Zeile zu LISTen. Ist dort die logische Filenummer hinter dem OPEN-Befehl »0«, so ist sie durch eine andere Zahl zu ersetzen. Vorsicht! Alle Ein-/Ausgabe-Befehle, die sich auf diesen OPEN-Befehl beziehen, müssen ebenfalls entsprechend geändert werden.

Steht in der geLISTeten Zeile kein OPEN-, sondern ein INPUT#- oder GET#-Befehl, dann ist versucht worden, aus einer Datei etwas zu lesen, die nur zum Schreiben geöffnet wurde (siehe Beispiel). Sollte dies eingetreten sein, ist zuerst zu überprüfen, ob der INPUT#- oder GET#-Befehl an der entsprechenden Stelle richtig ist. Wenn der Befehl stimmt, ist eine falsche Filenummer gewählt worden, oder die fehlerhafte Zeile wurde mit dem GOTO- beziehungsweise GOSUB-Befehl angesprungen, ohne zuvor die geöffnete Datei zum Schreiben zu schließen.

?NOT OUTPUT FILE:

Es wurde versucht, mit dem Basic-Befehl PRINT# etwas in eine Datei zu schreiben, die nur zum Lesen geöffnet wurde. Auch dieser Fehler bezieht sich nur auf das Arbeiten mit Datasette. Beispiel:

```
10 OPEN 1,1,0,"TEST":REM KASSETTENDATEI ZUM LESEN
ÖFFNEN. ES MUSS EINE DATEI MIT DEM NAMEN TEST
EXISTIEREN.
20 A$="ABC":PRINT#1,A$:REM DIESE ZEILE ERZEUGT
DEN FEHLER
30 CLOSE1:REM KANAL SCHLIESSEN
```

Die gleiche Fehlermeldung erscheint auch dann, wenn man die Tastatur (Geräteadresse 0) mit dem Befehl »OPEN 1,0« zum Lesen öffnet und versucht, mit dem Befehl »PRINT#1,"ABC"« etwas zum Kanal mit der Filenummer »1« zu schicken. Das heißt, die Tastatur läßt sich nicht als Ausgabegerät verwenden.

Tritt der Fehler in einem Programm auf, ist er genauso zu beseitigen wie der »NOT INPUT FILE«-Error.

?OUT OF DATA:

Die Fehlermeldung erscheint, wenn das Programm auf den Basic-Befehl READ stößt und entweder keine DATAs vorhanden sind oder schon alle gelesen wurden. Beispiel:

```
READ A$ <RETURN>
```

Wird der Befehl im Direktmodus eingegeben, erscheint der »OUT OF DATA«-Error, weil keine DATAs vorhanden sind. Meldet der Computer diesen Fehler, sind entweder DATAs vergessen worden einzugeben, oder die Schleifenvariable innerhalb einer FOR..NEXT-Schleife zählt zu weit. Beispiel:

```
10 FOR I=0 TO 10
20 READ A
30 NEXT I
40 END
50 DATA 1,2,3,4,5
```

Die Schleifenvariable »I« zählt von 0 bis 10, erwartet also 11 DATAs. Es sind aber nur fünf DATAs vorhanden. Folglich meldet der Computer einen »OUT OF DATA«-Error.

Wie der Fehler beseitigt wird, dürfte damit wohl auch geklärt sein. Hinweis: Der Fehler tritt zwar in der Zeile auf, deren Zeilennummer hinter der Fehlermeldung ausgegeben wird, befindet sich aber niemals in dieser Zeile, sondern immer in der Zeile, die die entsprechenden DATAs enthält.

?OUT OF MEMORY:

Dieser Fehler kann verschiedene Ursachen haben:

1) Der Speicherplatz im Computer reicht nicht für das Programm und die Variablen. Häufig tritt der Fehler auf, wenn DIM-Anweisungen zu groß gewählt wurden (zum Beispiel »DIM A(10000)«.

2) Wenn Maschinenprogramme geladen wurden, deren Startadresse über dem freien Basic-Bereich liegen (zum Beispiel SMON). In einem solchen Fall schafft der Befehl NEW Abhilfe, nachdem das Programm geladen wurde.

3) Verschachtelungstiefen bei FOR...NEXT-Schleifen (maximal 10) oder Unterprogrammaufrufen, mit dem Befehl GOSUB (maximal 24) wurden überschritten. Die häufigste Fehlerursache entsteht dadurch, daß in einem Unterprogramm der Befehl RETURN fehlt.

Wurde die Verschachtelungstiefe überschritten, muß das Programm anders aufgebaut werden. FOR...NEXT-Schleifen lassen sich zum Beispiel durch IF-Abfragen ersetzen. Beispiel

```
10 FOR I=0 TO 10
20 A=A+1
30 NEXT I
```

ist identisch mit

```
10 A=A+1
20 I=I+1
30 IF I <>10 THEN 10
```

?OVERFLOW:

Entweder liegt das Ergebnis oder ein Zwischenergebnis eines Ausdrucks außerhalb des zulässigen Zahlenbereichs. Jedes Ergebnis muß zwischen »+/-1.37E38« liegen. Beispiel:

```
PRINT 9*2E37/100 <RETURN>
```

Wird dieses Beispiel eingegeben, erscheint der »OVERFLOW«-Error. Formt man das Beispiel um, so läßt sich dieser Ausdruck durchaus berechnen:

```
PRINT 9/100*2E37
```

Der Fehler ist dadurch entstanden, daß das Zwischenergebnis »9*2E37« den zulässigen Zahlenbereich überschritten hat. Im Falle eines »OVERFLOW«-Errors ist also ein Ausdruck so abzuändern, daß kein Zwischenergebnis den zulässigen Wert überschreitet.

?REDIM'D ARRAY:

Das Programm trifft zweimal auf dieselbe DIM-Anweisung. Eine Feldvariable (zum Beispiel A(20)) darf in einem Programm nur ein einziges Mal dimensioniert werden. Beispiel:

```
10 DIM A(20)
20 GOTO 10
```

Es tritt ein »?REDIM'D ARRAY ERROR IN 10« auf. Dieser Fehler läßt sich meistens dadurch vermeiden, daß alle benötigten DIM-Anweisungen in die erste Programmzeile verlegt werden. Weiterhin sollte darauf geachtet werden, daß auf alle möglichen DIM-Zeilen kein GOTO-Sprung erfolgt. Wenn sich dies nicht vermeiden läßt, kann man ja ein CLR: vor den DIM-Befehl setzen. Dann werden allerdings auch alle anderen Variablen gelöscht. Beispiel:

```
10 CLR : DIM A(20)
20 GOTO 10
```

Diesmal tritt kein Fehler auf.

?REDO FROM START:

Eine INPUT-Anweisung erwartet die Eingabe einer Zahl; es wurden jedoch Buchstaben oder Zeichen eingegeben. Die ganze INPUT-Anweisung wird noch einmal bearbeitet.

```
10 INPUT A
20 PRINT A
```

Starten Sie das Programmm mit »RUN« und geben Sie zum Beispiel ein »X« ein (plus »RETURN«-Taste). Der Computer wird dies mit der beschriebenen Fehlermeldung quittieren und die INPUT-Anweisung noch einmal ausführen. Es dürfen nur Zahlen eingegeben werden! Ausnahme ist der Buchstabe »E« zur Kennzeichnung von Exponential-Zahlen (zum Beispiel 1.5E+3, also 1,5mal 10 hoch 3)

Tip: Überprüfen Sie Ihre Eingabe noch einmal sorgfältig auf unerlaubte Zeichen.

?RETURN WITHOUT GOSUB:

Der Computer trifft auf ein »RETURN«, ohne daß zuvor ein »GOSUB« aufgetreten ist. Beispiel:

```
10 RETURN
```

Wenn Sie das Programm starten, tritt sofort der genannte Fehler auf. Aber: Es kann auch sein, daß zwar eine GOSUB-Anweisung auftritt, aber der Computer einen zweiten RETURN-Befehl findet. Häufigste Ursache: Ein Programm mit einem oder mehreren Unterprogrammen hat nach Beendigung des Hauptteils keine END-Anweisung und »läuft« deshalb unbeabsichtigt in die erste Unterroutine. An deren Ende trifft der C64 dann auf ein RETURN, ohne daß jedoch ein GOSUB gegeben wurde. Beispiel:

```
10 REM HAUPTPROGRAMM
20 GOSUB 100
30 REM HAUPTPROGRAMMENDE
100 REM UNTERPROGRAMM
110 RETURN
```

Haben Sie den Fehler entdeckt? Zuerst wird in Zeile 20 die Unterroutine ab Zeile 100 korrekt mit GOSUB angesprungen und mit RETURN beendet. Dann jedoch bearbeitet der C64 Zeile 30, Zeile 100 und dann Zeile 110! Abhilfe: Am Ende des Hauptteils eine END-Anweisung. Beispiel:

```
40 END
```

?STRING TOO LONG ERROR:

a) Eine Stringvariable enthält mehr als die erlaubten 255 Zeichen, oder

b) Es wurde mittels INPUT# eine Zeichenkette eingelesen, die mehr als 89 (!) Zeichen enthält. Beispiele:

a)

```
10 A$="X"
20 FOR I=1 TO 500
30 A$=A$+"X"
40 PRINT LEN(A$)
50 NEXT I
```

In Zeile 30 wird der String A$ immer um ein Zeichen erweitert. Zeile 40 informiert über die aktuelle Länge des Strings. Sobald er mehr als 255 Zeichen enthält, wird ein »?STRING TOO LONG ERROR IN 30« ausgegeben.

b)

```
10 OPEN 1,8,1,"TEST,S,W"
20 FOR I=1 TO 100
30 PRINT#1,"X";
40 NEXT I
50 CLOSE 1
60 OPEN 1,8,0,"TEST,S,R"
70 INPUT#1,A$
80 PRINT A$
90 CLOSE 1
```

Der erste Programmteil (Zeilen 10 bis 50) eröffnet eine Datei auf Diskette und schreibt 100 Zeichen direkt hintereinander in diese Datei. Direkt hintereinander deshalb, weil in Zeile 30 nach dem PRINT#-Befehl ein »;« steht.

Der zweite Programmteil (Zeilen 60 bis 80) versucht nun, diese 100 Zeichen auf einmal in die Variable A$ einzulesen (was natürlich nicht funktioniert).

Um diesen Fehler zu verstehen, muß man sich die Funktionsweise des INPUT#-Befehls vor Augen führen: INPUT# liest solange Zeichen in die angegebene Variable, bis in der Datei das erste »carriage return« (CHR$(13)) folgt. Dieses Zeichen, kurz »cr« genannt, wird beim Schreiben von Daten in eine Datei immer genau dann gesendet, wenn der PRINT#-Befehl nicht mit einem »,« oder »;« endet. Im Beispielprogramm wird in Zeile 30 durch das »;« nach dem PRINT#-Befehl das Senden des »cr« verhin-

dert. Deshalb meldet der Computer beim INPUT# in Zeile 70 einen STRING TOO LONG ERROR, sobald er merkt, daß nach dem 89. Zeichen immer noch kein »cr« in der Datei aufgetaucht ist. Die Daten werden nämlich nicht sofort in die Variable A$ eingelesen, sondern erst in einen Zwischenpuffer; und dieser kann nicht mehr als 89 Zeichen aufnehmen. Um unser Programm doch noch lauffähig zu machen, müßte man den INPUT#-Befehl durch eine FOR-NEXT-Schleife mit einer GET#-Anweisung ersetzen (diese liest immer nur ein einziges Zeichen). Also:

```
70 FOR I=1 TO 100 : GET#1,B$ : A$=A$+B$ : NEXT I
```

Wenn in einem Programm ein STRING TOO LONG ERROR in einer Zeile mit einer INPUT#-Anweisung auftritt, ist der Fehler also nicht in dieser Zeile, sondern in der Routine zu suchen, die die Daten in die Datei geschrieben hat. Diese muß vor dem Senden eines Strings über die LEN-Funktion erst einmal testen, ob er die vorgeschriebene Länge von 89 Zeichen nicht überschreitet.

?SYNTAX ERROR:

Kann mehrere Gründe haben:

a) Generell ist die Syntax eines Befehls, also seine Schreibweise, von Ihnen falsch eingegeben worden.

b) In einer DATA-Zeile steht ein Zeichen oder eine Zeichenfolge, obwohl der READ-Befehl eine Zahl erwartet.

c) Das erste Zeichen des Basic-Speichers ist nicht Null.

Beispiele:

a) Allgemein tritt der Syntax-Error auf, wenn der Basic-Interpreter einen Befehl nicht versteht, also wenn Sie sich beim Schreiben eines Programms vertippt haben. Überprüfen Sie daher die Zeile, in der der Fehler auftrat, noch einmal sorgfältig, ob Sie die Befehle alle richtig geschrieben oder irgendein Zeichen vergessen haben! Häufigster Tippfehler aller Programmierer: »RUM« statt »RUN«.

Dieser Fehler tritt auch auf, wenn in einer Variablenzuweisung nicht alle Klammern geschlossen wurden. Beispiel: `A=(2+(3*4)`. Es fehlt eine »Klammer zu«. Oder: `A=2+(3*4))`. Eine Klammer wurde zu viel geschlossen.

b)

```
10 READ A,B,C
20 DATA 67, Z, 20
```

Es tritt ein Syntax-Error in Zeile 20 (!) auf, da die READ-Anweisung in Zeile 10 drei Zahlen als DATA-Werte erwartet. Die »zweite Zahl« ist jedoch ein Buchstabe.

Tip: Wenn Sie in DATA-Zeilen Zahlen und Buchstaben zusammen verwenden möchten und es tritt dieser Fehler auf, dann überprüfen Sie, ob Sie die Werte auch in der richtigen Reihenfolge auslesen.

c) Besonders tückisch: Wenn aus irgendeinem Grund der Inhalt der ersten Basic-Adresse nicht Null ist, so tritt bei der Eingabe von »RUN« oder »NEW« ein Syntax-Error auf. Dies ist meistens der Fall, wenn Sie den Basic-Start (Adressen 43 und 44) verändert haben, zum Beispiel, um eine Basic-Erweiterung zu verwenden.

Abhilfe: `POKE(PEEK(43)+PEEK(44)*256—1),0`

?TOO MANY FILES:

Es wurden mehr als die maximal erlaubten zehn Dateien (über einen OPEN-Befehl) eröffnet. Beispiel:

```
10 OPEN 1,3
20 OPEN 2,3
30 OPEN 3,3
40 OPEN 4,3
50 OPEN 5,3
60 OPEN 6,3
70 OPEN 7,3
80 OPEN 8,3
90 OPEN 9,3
100 OPEN 10,3
110 OPEN 11,3
```

Sicherlich, dies ist ein extremes Beispiel, das im »Alltagsbetrieb« nie auftreten wird, aber ebenso ist es mit dieser Fehlermeldung. Sollte also bei Ihnen dieser Fehler auftreten, so sollten Sie sich ernsthaft Gedanken darüber machen, ob Sie wirklich so viele offene Datenkanäle benötigen.

?TYPE MISMATCH:

Tritt auf, wenn versucht wird, eine Zeichenkette in einer numerischen Variablen abzulegen. Beispiele:

`A="TEST"` oder `A=A$`

Untersuchen Sie die Zeile, in der der Fehler aufgetreten ist. Meistens wurde nur das $-Zeichen vergessen.

?UNDEF'D FUNCTION:

Eine mathematische Funktion wurde angesprochen, ohne vorher definiert worden zu sein. Beispiel:

```
10 A=FN X(2)
```

Wenn Sie trotzdem der Meinung sind, die entsprechende Funktion definiert zu haben, so überprüfen Sie den verwendeten Funktionsnamen noch einmal. Zur Veranschaulichung hier noch einmal die Syntax der FN-Anweisung:

Definieren: DEF FN name(variable)=funktion

Aufruf: variable=FN name(wert oder variable)

Ein korrektes Beispiel:

```
10 DEF FN TEST(X)=X*2
20 A=FN TEST(2)
30 PRINT A
```

?UNDEF'D STATEMENT:

a) Über `RUN`, `GOTO` oder `GOSUB` wurde eine Zeile angesprochen, die nicht existiert.

b) Es wurde versucht, eine Variable als Sprungziel zu verwenden. Beispiel:

a)

```
10 GOTO 100
20 END
```

Dieser Fehler ist äußerst einfach zu beheben:

Zeile, in der der Fehler auftrat, listen, und überprüfen, ob die angesprochene Zeile wirklich vorhanden ist. Dann die GOTO- oder GOSUB-Anweisung entsprechend ändern.

b)

```
10 A=100 : GOTO A
100 END
```

Obwohl die Zeile 100 existiert, tritt der Fehler auf. Es darf nicht über eine Variable zu einer bestimmten Zeile gesprungen werden. Es gibt aber die Möglichkeit, über einen ON-GOTO-Befehl einen berechneten Sprung auszuführen:

```
10 INPUT "SPRUNG NACH ZEILE?";A
20 A=A/100 : ON A GOTO 100,200,300
100 PRINT "ZEILE 100" : END
200 PRINT "ZEILE 200" : END
300 PRINT "ZEILE 300" : END
```

Beachten Sie aber, daß der ON-GOTO-Befehl wie folgt arbeitet:

Wenn A=1, dann GOTO 100, wenn A=2, dann GOTO 200, ...

Deshalb muß die eingegebene Zahl auch erst in dieses Format umgerechnet werden (`A=A/100`).

?VERIFY ERROR:

Durch ein `VERIFY`-Kommando wurde festgestellt, daß das Programm im Speicher mit dem auf Datasette beziehungsweise Diskette nicht übereinstimmt. Deutet bei Datasettenbetrieb meistens auf ein fehlerhaftes Band hin. Verwenden Sie dann eine neue Kassette.

Diese Fehlermeldung ist bei Diskettenbetrieb äußerst unwahrscheinlich, da die Floppy beim Speichern eines Programms automatisch einen Verify durchführt. Eventuell schadhafte Disketten werden daher schon frühzeitig erkannt. Generell kann man sagen, daß beim Arbeiten mit einer Floppy das »Verifizieren« entfallen kann. Nützlich dagegen ist es bei der Programmentwicklung, wenn man nicht sicher ist, ob man die aktuelle Version schon gespeichert hat. Hier bringt ein Verify die Antwort.

(Achim Hübner/ef)

Solange der Computer noch brav seine Fehlermeldung ausgibt, hat man ja Glück gehabt. Kritisch wird die Situation dann, wenn auf dem Bildschirm so ein eigenartiges Gemisch undefinierbarer Zeichen erscheint und sich der Computer weder durch Betätigen aller erreichbaren Tasten noch durch gutes Zureden beeinflussen läßt.

Wer letzteres schon einmal erlebt hat, wird unserer ersten Regel sicher zustimmen: Jedes Programm sollte vor dem Start unbedingt mit »SAVE« gesichert werden.

Das Retten des mühevoll erstellten Programms ist allerdings noch nicht die Lösung, denn der Fehler ist nach wie vor vorhanden. Also beginnt die Fehlersuche. Diesen Vorgang bezeichnet man auch als »Debugging«. Das Wort ist von der englischen Bezeichnung »Bug« abgeleitet und bedeutet eigentlich »entwanzen«, wobei mit den »Wanzen« die Fehler gemeint sind, die sich überall im Programm verstecken. In der amerikanischen Umgangssprache hat sich das Wort »Debugging« ganz allgemein für das Suchen von Fehlern eingebürgert.

Ganz grob kann man zwischen zwei Arten von Fehlern unterscheiden. Einerseits gibt es die logischen Fehler, die mit schöner Regelmäßigkeit in der Entwicklungsphase eines Programms auftauchen, weil man dem Computer noch nicht genau genug gesagt hat, was er nun eigentlich tun soll. Diese Art von Fehlern erkennt man meist daran, daß das Programm anstandslos läuft, aber nicht immer die gewünschten Ergebnisse produziert. Die zweite Art von Fehlern tritt vorwiegend auf, wenn man ein Programm von einem fremden Listing oder von den eigenen Aufzeichnungen abtippt: Es handelt sich dann zumeist um schlichte Tippfehler oder um Fehler, die auf schlechte Lesbarkeit der Vorlage beruhen. Wir wollen uns im folgenden nur mit der zweiten Art von Fehlern beschäftigen.

Wie geht man nun zweckmäßigerweise vor, um alle Fehler zu finden, ohne das gesamte Programm von Anfang bis Ende mit der Vorlage Zeichen für Zeichen vergleichen zu müssen? Nun, eine allgemeingültige Methode, die für alle Arten von Programmen anwendbar wäre, gibt es leider nicht. Dennoch ist ein systematisches Vorgehen angebracht.

Der Computer hilft bei der Fehlersuche

Zunächst sollten wir uns darüber klarwerden, inwieweit uns der Computer selbst bei der Suche nach Fehlern helfen kann. Als erstes kommen einem dabei natürlich die Fehlermeldungen in den Sinn, die beim Commodore-Basic ja erfreulicherweise im Klartext erfolgen und recht vielfältig sind. Was die einzelnen Fehlermeldungen bedeuten und jeweils ein paar Tips zur Fehlersuche finden Sie in dem Artikel »Mein Computer versteht mich nicht« auf der Seite 129. Wir wollen uns hier mit den »härteren Fällen« beschäftigen.

Was soll man beispielsweise tun, wenn der Computer gar keine Fehlermeldung ausgibt, sondern sich nach »RUN« sang- und klanglos verabschiedet und auf keinen Tastendruck reagiert.

Für solche Fälle bietet Basic zwei spezielle Befehle, die man immer dann einsetzen sollte, wenn man nicht genau weiß, an welcher Stelle des Programms der Fehler steckt. Gemeint sind die Basic-Befehle STOP und CONT. Wenn der Computer beim Abarbeiten des Programms auf den Befehl STOP stößt, unterbricht er die Programmausführung und gibt eine Meldung »BREAK IN nnn« aus, wobei nnn die Zeilennummer ist, in der er die STOP-Anweisung gefunden hat. Alle Variablen und auch der Stackpointer bleiben dabei erhalten, so daß die STOP-Anweisung auch in Unterprogrammen oder innerhalb von FOR-NEXT-Schleifen verwendet werden kann.

Nach einem so erzwungenen Programmstopp kann man sich im Direktmodus mit dem PRINT-Befehl über die Werte aller wichtigen Variablen informieren und sogar mit LIST einzelne Programmteile anschauen. Danach gibt man den CONT-Befehl, und das Programm wird ganz normal fortgesetzt. Wenn es zu Testzwecken notwendig erscheint, kann man während eines Stopps auch im Direktmodus Variablenwerte verändern oder FOR-NEXT-Schleifen verwenden. Allerdings dürfen weder neue Programmzeilen eingegeben noch alte gelöscht oder verändert werden, da dadurch alle Variablen gelöscht werden und CONT anschließend nicht mehr möglich ist.

Es ist empfehlenswert, an mehreren kritischen Stellen des Programms STOP-Befehle einzufügen, um den Programmlauf verfolgen zu können und den Fehler immer mehr einzugrenzen. Kritische Stellen sind generell und ohne Ausnahme alle SYS- und USR-Aufrufe, desgleichen alle POKE-Befehle, über deren Bedeutung man sich nicht hundertprozentig im klaren ist. Im Zweifelsfalle sollte man auch nach jedem GOSUB im Programm zunächst einen STOP-Befehl einbauen, um sicherzugehen, daß das Unterprogramm auch wieder auf normalem Wege verlassen wird.

Jedesmal, wenn man durch davor oder danach plazierte STOP-Befehle festgestellt hat, daß beispielsweise ein SYS-Befehl ordnungsgemäß abgearbeitet wird, kann man die STOPs natürlich wieder entfernen, um einen flüssigeren Programmablauf zu erreichen. Es empfiehlt sich, alle eingefügten STOP-Befehle auf einem Zettel zu notieren, um die Übersicht zu behalten. Schließlich dienen die Befehle

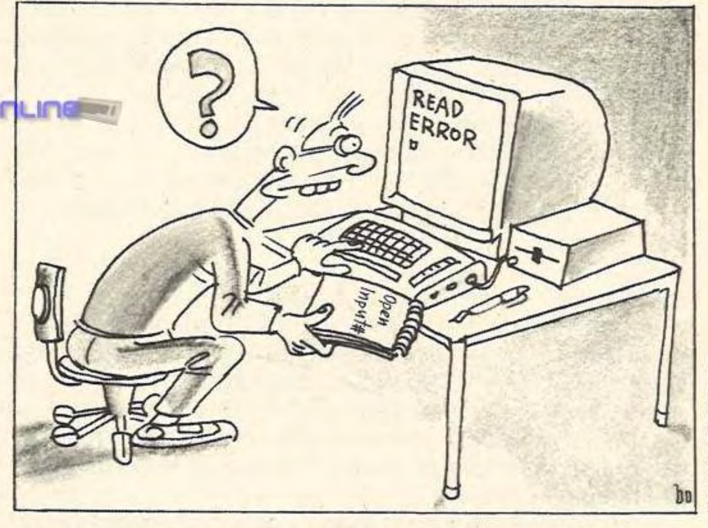


Illustration Rolf Boyke

nur der Fehlersuche und müssen irgendwann alle wieder entfernt werden.

In vielen Fällen kann man STOP-Anweisungen durch einfache PRINT-Anweisungen ersetzen. Das hat den Vorteil, daß keine Programmunterbrechung stattfindet und man nicht jedesmal CONT eintippen muß. Außerdem kann man in PRINT-Anweisungen auch zusätzliche Informationen geben, zum Beispiel Variablenwerte ausdrucken oder direkt auf ein spezielles Problem aufmerksam machen. Diese PRINT-Anweisungen sollten aber in irgendeiner Weise von den normalen Bildschirmausgaben unterscheidbar sein. Zum Beispiel kann man jede PRINT-



Anweisung zur Fehlersuche mit fünf Sternchen oder fünf Pluszeichen beginnen lassen.

Will man sich mehrere Variable während des Programmlaufs ausdrucken lassen, sind kleine Unterprogramme recht hilfreich, die in einem freien Zeilenbereich geschrieben werden und die alle benötigten Ausgaben durchführen. Am Ende solcher Unterprogramme sollte eine GET-Scheife stehen, die das Programm auf Tastendruck weiterlaufen läßt. Statt langer PRINT-Listen braucht man so nur einen GOSUB-Aufruf überall dort im Programm einfügen, wo dies sinnvoll erscheint.

Das Arbeiten mit STOP und CONT mag manchem Computerneuling etwas ungewohnt erscheinen, aber es ist jedenfalls ein recht sicheres Mittel, einem immer wieder abstürzenden Programm auf die Schliche zu kommen.

Wenn der C64 nur noch »Bahnhof« versteht...

Die häufigste Fehlermeldung ist sicherlich der ungeliebte »SYNTAX ERROR«. Solange der Computer nur Syntax-Fehler meldet, kann man noch von Glück reden. Es handelt sich dabei meistens um einfache Tippfehler, die nach Auflisten der entsprechenden Zeile leicht zu finden und zu korrigieren sind.

Häufige Fehler sind zum Beispiel fehlende oder überzählige Klammern und die Verwechslung ähnlicher Zeichen wie zum Beispiel »0« und »O«, »1« und »I« oder »8« und »B«. Sehr häufig ist auch die Verwechslung von Punkt und Komma, was sich besonders in DATA-Zeilen verhängnisvoll auswirken kann, wie wir nachher noch sehen werden. Wenn Sie also irgendwo einen »SYNTAX ERROR« gemeldet bekommen und in der fraglichen Zeile auf Anhieb keinen Fehler finden, dann gehen Sie zuerst die vorhin genannten Punkte durch.

Eine gute Hilfe ist es, mit dem Cursor die fehlerhafte Zeile Zeichen für Zeichen abzufahren und dabei mit der Vorlage zu vergleichen. Kommen in der Fehlerzeile viele Klammern vor, dann empfiehlt sich häufig das Anlegen einer Strichliste für öffnende und schließende Klammern. Die Anzahl muß innerhalb jeder Basic-Anweisung übereinstimmen. Aber bitte keine Klammern mitzählen, die in Ausführungszeichen stehen, diese haben mit der »Syntax« nichts zu tun.

Ab und zu kann es vorkommen, daß man bei aller Sorgfalt einen Syntaxfehler nicht findet, wie zum Beispiel in der folgenden Basic-Zeile:

```
10 OPEN 1,4:PRINT#1,"HALLO":CLOSE 1
```

Wenn der Computer hier dennoch einen Syntaxfehler meldet, dann kann das nur eine Ursache haben: Bei der Eingabe dieser Zeile wurden die Basic-Befehle in der bekannten Art und Weise abgekürzt, der zweite Befehl dabei aber als »?#1« eingegeben, was beim Auflisten wieder zu »PRINT#1« wird. Leider sind PRINT und PRINT# zwei völlig verschiedene Befehle, genauso wie INPUT und INPUT# oder GET und GET#. Also bitte bei der abgekürzten Eingabe von »PRINT#« unbedingt die im Handbuch angegebene Form <P><SHIFT R> statt <?><#> verwenden. Die normale PRINT-Routine weiß nämlich mit dem nachfolgenden »#« nichts anzufangen, und es kommt zu einer Fehlermeldung.

Eine ähnliche Situation kann sehr leicht bei der Verwendung langer Variablennamen auftreten. In einem Spielprogramm »Schiffe versenken« kann zum Beispiel folgende Zeile auftreten:

```
10 ANZAHLSCHIFFE = 12
```

In dieser Zeile wird unweigerlich ein »SYNTAX ERROR« auftreten, weil der Computer innerhalb des Variablennamens »ANZAHLSCHIFFE« das Basic-Schlüsselwort »IF« entdeckt und sich bei aller Anstrengung nicht erklären kann, was eine IF-Abfrage an dieser Stelle soll. Trotz aller Vorteile für die Übersichtlichkeit eines Programms sei daher an dieser Stelle von der Benutzung langer Variablennamen abgeraten.

Eine andere häufig auftretende Fehlermeldung ist vor allem von Anfängern gefürchtet, nämlich »OUT OF DATA ERROR«.

Was tun bei »OUT OF DATA«?

Gefürchtet ist dieser Fehler deswegen, weil die Zeilennummer, die der Computer zu dieser Fehlermeldung ausgibt, in den meisten Fällen keinen Hinweis darauf gibt, an welcher Stelle der eigentliche Fehler liegt. Listet man nämlich die als fehlerhaft angegebene Zeile am Bildschirm auf, so findet man dort nur den READ-Befehl, für den keine DATAs mehr vorhanden waren. Einen näheren Hinweis erhält man durch:

```
Print PEEK(63) + 256 * PEEK(64)
```

Dieser Befehl listet die Zeilennummer der zuletzt gelesenen DATA-Zeile. Ein typisches, wenn auch stark vereinfachtes Beispiel für das Auftreten von Fehlern im Zusammenhang mit DATA-Anweisungen, ist in Listing 1 gegeben. In den Zeilen 100 bis 170 wird eine Prüfsumme über den ersten DATA-Block gebildet, und nur dann, wenn diese Prüfsumme in Ordnung ist, wird in den nächsten Programmteil verzweigt, wo aus dem zweiten DATA-Block Zahlen gelesen und an den Anfang des Bildschirms gePOKEt werden und dort das Wort »COMMODORE« bilden sollen.

Das Programm enthällt nun einige Fehler in den DATA-Zeilen, die wir gemeinsam herausfinden wollen. Stellen wir uns einfach vor, wir hätten das Programm in Listing 1 aus einer Zeitschrift abgetippt und dabei einige Fehler in den DATA-Zeilen fabriziert. In Listing 2 sind zum Vergleich noch einmal die entsprechenden DATA-Zeilen des »Original«-Listings abgedruckt. Die Fehlersuche erscheint somit recht

hler auf der Spur

Diese Situation kennt wohl jeder: Da tippt man stundenlang ein Programm ab und lehnt sich nach dem letzten »RETURN« erleichtert zurück. Jetzt folgt das magische »RUN« – und nichts läuft.

einfach: Man vergleicht die paar DATAs in den beiden Listings und wird dann schon den Fehler finden. Für dieses kurze Testprogramm stimmt das natürlich auch. Aber stellen wir uns einmal vor, daß die beiden DATA-Blöcke insgesamt vielleicht über drei volle Listing-Seiten gehen und nicht nur über drei Zeilen wie in unserem Beispiel. Dann lohnt es sich mit Sicherheit, wenn man etwas systematischer an die Fehlersuche herangeht.

Zuerst starten wir unser Programm nach Listing 1 einmal ganz arglos mit RUN, nachdem wir es vorher gespeichert hatten. Der Programmablauf ist zu Anfang ganz wie erwartet: Der Bildschirm wird gelöscht, es erscheint die Meldung »S = 282« und darunter »OK«, dann jedoch erscheinen am oberen Bildschirmrand statt eines längeren Wortes nur die beiden Zeichen »%« und »C« und das Programm bricht mit der Meldung »? ILLEGAL QUANTITY ERROR IN 230« ab. Was ist hier geschehen?

Wenn wir uns Zeile 230 auflisten lassen, dann sehen wir:

```
230 POKE B+I,X
```

Da wir wissen, daß nur Zahlen zwischen 0 und 255 gePOKEt werden können, vermuten wir den Fehler beim Wert der Variablen »X«. Um unsere Vermutung zu bestätigen, fragen wir den Computer doch einmal ganz einfach nach dem Wert von X, indem wir eintippen:

```
PRINT X
```

Anschließend bestätigen wir mit der RETURN-Taste. Wir erhalten als Antwort den Wert 1513, der tatsächlich zu groß ist, um in eine Speicherzelle zu passen. Wir vergleichen den zweiten DATA-Block mit dem Original (Listing 2) und

Kein Verlaß auf Prüfsummen

stellen fest, daß wir bei der Eingabe das Komma zwischen den Zahlen 15 und 13 in Zeile 360 vergessen haben. Das ändern wir, indem wir das Komma nachträglich einfügen.

Dank dieses schnellen Erfolges bessert sich unsere Stimmung um einiges, was sich jedoch nach dem nächsten RUN sehr schnell wieder ändert. Zwar erscheint am Bildschirm zunächst ganz ordentlich die Prüfsumme des ersten DATA-Blockes und das dazugehörige »OK«, aber am oberen Bildschirmrand stimmt einiges noch nicht: Man liest dort die Zeichenfolge »%%COMMOORE« statt »COMMODORE«.

Auf den ersten Blick würde man vielleicht vermuten, daß der Fehler nur im zweiten DATA-Block stehen kann, weil das Lesen des ersten Blocks keine Fehlermeldung erzeugt und sogar die Prüfsummenbildung stimmt. Diese Überlegung ist nicht ganz schlüssig. Denn durch Bildung einer einfachen Prüfsumme werden Vertauschungsfehler und überflüssige oder fehlende Nullen nicht erkannt. Die fünf DATA-Zeilen in Listing 3 ergeben zum Beispiel alle die gleiche Prüfsumme.

Man sollte sich also nie blindlings auf Prüfsummen verlassen. Sie sind zwar oft nützlich, um Fehler in DATA-Zeilen festzustellen, man darf aber aus der Richtigkeit der Prüfsummenprobe niemals schließen, daß kein Fehler enthalten sein kann. Außerdem taucht eine weitere Schwierigkeit auf: Wenn man nur eine globale Prüfsumme über alle DATAs bildet, dann kann man zwar unter Umständen einen Fehler nachweisen, weiß aber immer noch nicht, wo er steckt. Da muß man dann schon zu anderen Mitteln greifen.

Um den Fehler aufzuspüren, können wir dem Computer einen großen Teil der Arbeit überlassen. Als erstes wollen wir feststellen, in welchem der beiden DATA-Blöcke der Fehler liegen könnte. Dazu veranlassen wir den Computer einfach, nur den ersten DATA-Block zu lesen, indem wir eine STOP-Anweisung hinter die erste FOR-NEXT-Schleife plazieren. Wir fügen also folgende Zeile ins Programm ein:

```
145 STOP
```

Wenn wir das Programm jetzt starten, erhalten wir die Meldung »BREAK IN 145«, die von unserem STOP-Befehl herrührt. Da aber das Programm bis Zeile 145 durchlaufen wurde, muß der erste DATA-Block an dieser Stelle vollständig gelesen worden sein. Die Variable »X« erhält natürlich immer noch den zuletzt gelesenen DATA-Wert. Wenn dieser Programmteil richtig gearbeitet hat, dann müßte X jetzt den Wert 0 haben, denn dies ist ja gerade der letzte DATA-Wert aus Block 1, wie man anhand von Listing 1 oder 2 unschwer erkennen kann. Das können wir einfach nachprüfen, indem wir den Computer nach dem Wert »X« fragen:

```
PRINT X
```

Zu unserem Erstaunen ist die Antwort aber nicht 0, sondern 12. Wir werfen wieder einen Blick auf Listing 1 und stellen fest, daß die Zahl 12 die vorletzte Zahl im ersten DATA-Block ist. Offenbar wurde eine Zahl zuwenig gelesen! Es ist nun verlockend, einfach den Endwert der ersten FOR-NEXT-Schleife um eins zu erhöhen, um alle Werte des ersten Blocks zu lesen. Doch halt, hier ist Vorsicht geboten. Viel wahrscheinlicher als ein Fehler in einer FOR-NEXT-Schleife ist ein Fehler innerhalb der DATA-Zeilen. Bei so vielen Zahlenangaben kann man sich schließlich leicht mal vertippen.

Betrachten wir das Problem einmal von der anderen Seite. Wenn die Anzahl der gelesenen X-Werte stimmt, das Programm aber trotzdem nur bis zum vorletzten DATA-Wert kommt, dann enthält Block 1 vielleicht einen DATA-Wert zuviel. Wir wollen also die DATAs in Block 1 ganz gezielt überprüfen. Dazu schreiben wir in einen freien Zeilenbereich, zum Beispiel ab Zeile 1000, das folgende kleine Unterprogramm:

```
1000 PRINT "I =" ; I , "X =" ; X
1010 GET A$ : IF A$ <>CHR$(32) THEN 1010
1020 RETURN
```

In die erste FOR-NEXT-Schleife fügen wir direkt hinter die READ-Anweisung einen Aufruf dieses Unterprogramms ein:

```
125 GOSUB 1000
```

Wenn wir das Programm nun laufen lassen, geschieht folgendes: Der Computer gelangt mit Zeile 110 in die Leseschleife. In Zeile 120 wird jeweils ein DATA-Element gelesen. Dann erfolgt mit der eingefügten Zeile 125 ein Sprung

Hilfsprogramm bei DATA-Fehlern

in das vorhin geschriebene Unterprogramm. Dieses Unterprogramm druckt den Wert der Zählvariablen »I« und den soeben gelesenen DATA-Wert »X« aus und wartet dann, bis die Leertaste betätigt wird. Dann kehrt das Unterprogramm zurück und die Schleife wird nach dem NEXT in Zeile 140 erneut durchlaufen.

Auf diese Weise erhält man auf dem Bildschirm eine übersichtliche Darstellung der gelesenen DATA-Werte, die man leicht mit der Vorlage vergleichen kann. Wenn wir das Programm jetzt starten, erhalten wir jeweils nach Drücken der Leertaste eine Bildschirmanzeige etwa in folgender Art:

```
I = 1          X = 12
I = 2          X = 33
I = 3          X = 11
```

und so weiter – bis schließlich das Ende von DATA-Zeile 310 erreicht wird:

```
I = 9          X = 18
I = 10         X = 0
```

Nanu? Das hatten wir eigentlich nicht erwartet. X=18 ist der letzte Wert in Zeile 310 und es sollte eigentlich der erste Wert aus der nächsten DATA-Zeile gelesen werden, nämlich X=11. Woher kommt dieser Wert Null bei I=10? Ein Vergleich von Zeile 310 in Listing 1 (abgetippt) mit Listing 2 (Original) führt uns auf des Rätsels Lösung. Offenbar haben wir beim Abtippen am Ende von Zeile 310 noch ein Komma gesetzt, was dort nicht hingehört. Ein Komma in einer DATA-Anweisung trennt für unseren C64 aber immer zwei Werte voneinander. Da er hinter dem letzten Komma nichts mehr findet, setzt er kurzerhand den Wert »0« dafür an.

Kontrolle ist besser

Damit haben wir den überzähligen DATA-Wert im ersten Block gefunden. Wir entfernen das Komma in Zeile 310 und löschen die Zeile 125 mit dem GOSUB-Befehl sowie die Zeile 145 mit dem nun nicht mehr benötigten STOP-Befehl.

Ein erneuter Probelauf des Programms schreibt die Zeichenfolge »COMMOORE« links oben auf den Bildschirm – und bringt die Fehlermeldung »OUT OFF DATA ERROR IN 220«. Nach Auflisten der Zeile 220 sehen wir leider nur:

```
220 READ X
```

Das bringt uns nicht sehr viel weiter. Die Fehlermeldung und das verstümmelte Wort »COMMODORE« am oberen Bildschirmrand deuten auf einen fehlenden DATA-Wert in Block 2 hin. Untersuchen wir also Block 2 einmal genauer. Das Unterprogramm zum Ausdrucken der Werte von »I« und »X« auf den Bildschirm befindet sich ab Zeile 1000 noch im Speicher. Wir brauchen daher nur einen entsprechenden GOSUB-Befehl in die zweite Leseschleife einzufügen, am besten gleich nach dem READ-Befehl, also in Zeile 225:

```
225 GOSUB 1000
```

Wir erhalten weder eine leicht zu überprüfende Liste aller DATA-Werte, diesmal aus Block 2. Bei I=4 fällt uns sofort etwas auf. Am Bildschirm erscheint nämlich:

```
I = 4                X = 15.4
```

Das ist die einzige Zahl mit Normalkommastelle, was bei dieser Art der Bildschirmausgabe sofort ins Auge sticht. Wir vergleichen den Wert mit der Eintragung ins Originallisting und sehen sofort den Fehler: Wir haben beim Abtippen irrtümlich einen Punkt statt eines Kommas eingegeben. Die Korrektur ist leicht ausgeführt.

Natürlich kann man nicht erwarten, daß sich alle Fehler so reibungslos lokalisieren lassen wie in unserem kleinen Beispiel. Gerade bei Fehlern in DATA-Zeilen kann die Suche sich bei längeren DATA-Blöcken um einiges schwieriger gestalten. Aber bei Programmen mit vielen DATA-Zeilen sind die hier beschriebenen Methoden zum Auffinden von versteckten Fehlern einfach unentbehrlich, wenn man einigermaßen schnell und sicher zum Ziel gelangen will.

(V. Everts/U. Jürgens/kn)

```
1 REM DATA-TEST
2 REM ---------
3 REM
4 REM
5 B=1024:REM BILDSCHIRM
6 REM
7 PRINT"□":REM CLEAR
8 PRINT:PRINT:PRINT:PRINT
9 REM
100 REM DATA-BLOCK 1 LESEN
105 REM
110 FOR I=1 TO 17
120 READ X
130 S=S+X
140 NEXT I
150 PRINT"S =";S
160 IF S<>282 THEN PRINT"PRUEFSUMMENFEHLER":END
170 PRINT"OK"
190 REM
195 REM
200 REM DATA-BLOCK 2 LESEN
205 REM
210 FOR I=0 TO 8
220 READ X
230 POKE B+I,X
240 NEXT I
250 END
290 REM
295 REM
300 REM DATA-BLOCK 1
305 REM
310 DATA 12,33,11,4,17,38,22,19,7,18,
320 DATA 11,41,15,19,3,12,0
345 REM
350 REM DATA-BLOCK 2
355 REM
360 DATA 3,1513,13,15.4,15,18,5

READY.
```

Listing 1. Achtung – fehlerhaftes Listing. In diesem Testprogramm sind in den DATA-Zeilen einige Fehler enthalten

```
290 REM
295 REM
300 REM DATA-BLOCK 1
305 REM
310 DATA 12,33,11,4,17,38,22,19,7,18
320 DATA 11,41,15,19,3,12,0
345 REM
350 REM DATA-BLOCK 2
355 REM
360 DATA 3,15,13,13,15,4,15,18,5

READY.
```

Listing 2. Hier nochmals der DATA-Block aus Listing 1, aber diesmal das fehlerfreie »Original«

```
100 DATA 12,13,14,15,16,17,0
200 DATA 13,12,14,15,16,17,0
300 DATA 13,12,14,15,16,17,0,0,0
400 DATA 23,2,14,15,16,17,0
500 DATA 23,2,14,15,16,,17

READY.
```

Listing 3. Die Bildung einer Prüfsumme ist kein zuverlässiges Mittel, um Fehler in DATA-Zeilen zu entdecken. Wie man leicht nachrechnen kann, ergibt jede der fünf DATA-Zeilen die gleiche Prüfsumme.

Viele Wege führen nach Rom - dieser Spruch könnte das Motto sowohl unseres Wettbewerbs als auch unserer Aufgabenstellung sein. Diese ergibt sich aus einer kleinen Geschichte, die in Italien spielt. Für die Lösung sollten Sie ein Basic-Programm schreiben. Wenn dieses Programm nicht nur die richtige Lösung bringt, sondern auch besonders schnell ist, haben Sie alle Chancen, zu den Gewinnern zu zählen.
Für das schnellste Programm mit der richtigen Lösung erhalten Sie drei Bücher, die zur Grundausstattung eines jeden C64-Besitzers gehören sollten (Bild 1):
F.Riemenschneider: C64/C128 - Alles über Maschinensprache
Commodore Sachbuchreihe: Alles über den C64
F.Müller: C64 für Insider
Der Gewinner des zweiten Preises erhält zwei dieser Bücher nach seiner Auswahl, für den dritten Preis können Sie sich eines der Bücher auswählen. Die Sieger-Listings werden wir allen Lesern vorstellen. Wer diesmal kein Glück hat: Die Knobel-Ecke wird fortgesetzt.



Die Aufgabe

Die Wettbewerbsbedingungen finden Sie am Ende dieses Artikels. Damit aber genug der Vorrede, kommen wir zur Aufgabe:

Die Entdeckung des Alarich-Schatzes

Im Jahr 1980 fuhren fünf Freunde nach Italien, um am Ufer des Busento nach dem verschollenen Schatz des Alarich zu suchen.

Sie gruben und suchten und suchten und gruben, bis sie endlich im Schweiße ihres Angesichts erfolgreich waren: Nachdem sie viel Geröll beiseite geräumt hatten, entdeckten sie eine verfallene Truhe, bis zum Rand mit Münzen gefüllt. Die Mühe hatte sich gelohnt.

Es war zwar erst früher Abend - der Stundenzeiger der Uhr zeigte auf die 7; von der Wühlarbeit des Tages waren sie jedoch so erschöpft, daß sie beschlossen, sich zunächst schlafen zu legen, um die gerechte Aufteilung des Fundes am nächsten Morgen vorzunehmen.

Sie fielen sofort in einen tiefen Schlaf, aber schon nach einer Stunde erwachte der erste, von Mißtrauen gepeinigt. Er beschloß, »aus Sicherheitsgründen« sich seinen fünften Teil zu nehmen. Um den Geist des Alarich zu beruhigen, warf er zunächst - entsprechend der Uhrzeit - acht Münzen in den Busento, teilte dann die Münzen in fünf gleiche Teile (was genau »aufging«), versteckte seinen Anteil in seinem Gepäck und legte sich wieder schlafen.

Eine Stunde später erwachte der zweite. »Ich bin zwar ein guter Mensch, aber die anderen dort - naja - ich nehme mir lieber meinen gerechten Teil«, sagte er zu sich und schritt zur Tat. Entsprechend zur Uhrzeit warf er zunächst neun Münzen in den Busento, teilte dann die Goldstücke in fünf gleiche Teile (was genau »aufging«), versteckte seinen vermeintlichen Anteil in seinem Gepäck und legte sich schlafen.

Wieder eine Stunde später erwachte der dritte. Er verfuhr ähnlich wie seine Vorgänger: In Anlehnung an die Uhrzeit warf er zehn Münzen in den Busento, teilte die Münzen in fünf gleiche Teile (was genau »aufging«), versteckte seinen Anteil und begab sich wieder zu Bett.

Nach abermals einer Stunde erwachte der vierte - auch er warf entsprechend der Uhrzeit elf Münzen in den Fluß, teilte dann die Goldmünzen in fünf Teile (was genau »aufging«), versteckte seinen Anteil im Gepäck, um sich dann beruhigt wieder schlafen zu legen.

Um Mitternacht erwachte der fünfte, der wie seine »Freunde« gemäß der Uhrzeit zwölf Münzen zur Beruhigung des Alarich-Geistes in den Busento warf. Er teilte die Münzen in fünf gleiche Teile (was, wie üblich, genau »aufging«), versteckte sein Fünftel und legte sich schlafen.

Nach Sonnenaufgang erwachten die Schatzgräber, machten alle ein ehrliches Gesicht und begannen mit der Aufteilung der restlichen Münzen, was wiederum genau »aufging«.

Wieder in der Heimat angekommen, erfuhr die Polizei





Bild 1. Gewinnen Sie bei unserem Wettbewerb: Bücher, die für jeden C64-Besitzer unentbehrlich sind.



Die Knobel-Ecke 1

durch Indiskretionen von der Schatzräuberei, und wegen der Geschwätzigkeit der Ehefrauen der fünf Freunde konnte die »Verteilungsprozedur« exakt ermittelt werden.

Nur die Anzahl der gefundenen Goldmünzen blieb unbekannt, und keiner der Beteiligten wollte sich diesbezüglich äußern. Die Behörden suchen bis heute nach einem Fachmann, der mit Hilfe seines Computers berechnen kann, wie viele Goldstücke die fünf Freunde mindestens gefunden haben müssen.

An dieser Stelle sind Sie gefragt. Sind Sie der Fachmann (die Fachfrau), um mit dem C64 diese Aufgabe zu lösen?

Da bei unserem Wettbewerb eine optimale Programmierung, das heißt eine hohe Geschwindigkeit im Vordergrund steht, geben wir Ihnen Hinweise zur Lösung der Aufgabe.

Beginnen Sie bei Ihren Überlegungen mit dem Ende der Verteilungsprozedur. Die Zahl der Münzen, die am Morgen noch vorhanden war, ließ sich genau durch fünf teilen. Die kleinste ganzzahlige Anzahl (jede Verteilung ließ sich durchführen, ohne daß ein Rest übrigblieb) ist 5. Rechnen Sie hoch auf die Anzahl der Münzen, die der fünfte um Mitternacht verteilte. Zwei Punkte müssen bei jeder neuen Verteilung berücksichtigt werden:

a) Jeder der Freunde wirft je nach Uhrzeit eine bestimmte Zahl Münzen in den Fluß.

b) Jede Verteilung erfolgt so, daß kein Rest übrigbleibt. Die Anzahl der Münzen ist immer ganzzahlig.

Mit einer Beispielrechnung möchten wir Ihnen einen möglichen Lösungsweg zeigen. Es bleibt Ihnen überlassen, ob Sie ähnlich vorgehen oder einen ganz anderen Weg finden, effektiver ist.

Angenommen, die Zahl der Münzen am Morgen ist genau 20. Diese Zahl läßt sich zumindest ohne Rest durch fünf teilen.

Diese Anzahl der Münzen ist übriggeblieben, nachdem der letzte um Mitternacht seine Verteilung erledigt hat. Nennen wir nun die Zahl der Münzen, die er vor der Verteilung vorfand, beispielsweise mit X. Von dieser Zahl wirft er zwölf Münzen in den Fluß. Anschließend nimmt er sich von diesem Rest genau ein Fünftel fort. Der Rest beträgt dann, wie oben angenommen, 20. Versuchen wir das einmal in eine Formel zu fassen:

$$20 = x - 12 - \frac{x-12}{5}$$

Als Ergebnis für den Wert X erhalten wir, wenn die Formel nach X aufgelöst wird, genau 37. Prüfen wir: Zwölf Münzen in den Fluß, bleiben 25, ein Fünftel abgezogen, bleiben 20, wie im Beispiel angenommen. Ganzzahlig ist der Wert auch.

Weiter hochrechnen wollen wir nicht, die Lösung sollen Sie finden. Mit unserem »Arbeitslisting«, das wir für die Aufgabe programmiert haben, erhalten wir nach 13 Sekunden die Lösung auf dem Bildschirm. Übrigens - diese Zeit, die Sie mit Sicherheit unterbieten werden, beweist die hohe Leistungsfähigkeit des C64 mit seinem eingebauten Basic. Auf einem Personal Computer dauerte ein vergleichbares Cobol-Programm mehrere Minuten.

Um gleiche Voraussetzungen für alle Teilnehmer zu gewährleisten, verwenden Sie für Ihr Basic-Programm nur das Basic 2.0 des C64. Routinen in Maschinensprache, die Sie zur Beschleunigung einsetzen könnten, sind nicht erlaubt, geschickte Programmierung ist gefragt.

Die erste Zeile in Ihrem Programm muß lauten:

```
10 TI$="000000"
```

Damit wird - für alle Teilnehmer gleich - die Zeit auf Null gesetzt. Welche Zeilennummer Sie verwenden, ist Ihnen überlassen.

Die letzte Zeile Ihres Programms sollte lauten:

```
500 PRINT TI$
```

Das Programm muß vor der Ausgabe der abgelaufenen Zeit folgende Informationen auf dem Bildschirm anzeigen:

a) Aus welcher Mindestanzahl von Münzen bestand der Schatz ursprünglich?

b) Wie viele Münzen waren morgens bei der Verteilung noch vorhanden?

Schicken Sie Ihr Programm (Diskette und/oder Listingausdruck) bis zum 30. April 1989 an folgende Adresse:

Markt & Technik Verlag AG
Redaktion Sonderhefte
Stichwort: Knobel-Ecke (1)
Hans-Pinsel-Str. 2
8013 Haar bei München

Viel Erfolg beim Knobeln und Programmieren! (ef)

Ordnen Sie das Chaos

Kein »Virus«, sondern ein Programm hat einige unserer Basic-Listings völlig durcheinandergebracht. Die Zeilennummern stimmen nicht, die Programme laufen nicht mehr: Nutzen Sie Ihre Basic-Kenntnisse, erstellen Sie aus dem Chaos ein lauffähiges Programm.

Es gibt viele Möglichkeiten, die logischen Abläufe eines Basic-Programms nachzuvollziehen. Wir stellen Ihnen hier eine davon vor, die den Freunden logischer Denkspiele gefallen wird.

Wenn Sie sich unsere drei Beispiele (Listing 1 bis 3) genauer anschauen, werden Sie schnell feststellen, daß keines dieser Listings in der abgedruckten Form einen Sinn ergibt. Die Zeilennummern der Programme beginnen jeweils bei 1080 (die REM-Zeilen vorher sind nur Vorspann), die Zeilen selbst sind kunterbunt durcheinander. Daher finden Sie in den Listings beispielsweise Sprünge auf nicht vorhandene Zeilen.

Dennoch verbergen sich hinter den drei Listings kleine Programme, die – richtig geordnet und mit neuer Zeilennummer versehen – kurze Anwendungen darstellen.

Um die richtige Reihenfolge der Zeilennummern wiederherzustellen, sollten Sie bei jedem Listing einige Punkte beachten:

Die Zeilen 1000 bis 1070 haben für die eigentlichen Programme keine Bedeutung, sie bleiben unberücksichtigt. Jedes (korrekte) Programm beginnt mit der Zeilennummer 100, der Zeilenabstand beträgt 10.

Die Sprünge auf andere Zeilen in unseren Beispiellistings beziehen sich immer auf die korrekte Programmversion. Der Basic-Inhalt der einzelnen Zeilen darf also nicht verändert werden, nur die Reihenfolge und die Zeilennummern.

Dies ist eine kleine Hilfe, um bei Verwendung der korrekten Zeilennummern das Programm in seine ursprüngliche Form zu bringen.

Einige Hinweise über die Funktion der Beispiele wollen wir Ihnen noch geben:

Listing 1:

Das Programm ist eine Eingaberoutine für Zahlen, die auch Zehner-Potenzen akzeptiert. Andere Eingaben als Zahlen werden nicht akzeptiert.

```
1000 REM <><><><><><><><><><><><><><><><>    <084>
1010 REM <>                          <>    <016>
1020 REM <>     KNOBELN MIT BASIC    <>    <249>
1030 REM <>                          <>    <038>
1040 REM <> (C) BY DIRK NEUMEISTER   <>    <212>
1050 REM <>                          <>    <058>
1060 REM <><><><><><><><><><><><><><><><>    <146>
1070 END :                                 <009>
1080 IF EI$=CHR$(20)THEN LY=LEN(IN$):IF LY
     >0 THEN IN$=LEFT$(IN$,LY-1):PRINT"{SP
     ACE,2LEFT,SPACE,LEFT}";               <095>
1090 IF ASC(EI$)>57 OR ASC(EI$)<48 THEN 11
     0                                     <018>
1100 GET EI$:IF EI$=""THEN 110             <227>
1110 IF EI$="E"THEN 190                    <212>
1120 IN$=IN$+EI$:PRINT EI$;:GOTO 110       <233>
1130 POKE 204,0:IN$="":PRINT"]";           <093>
1140 IF EI$=CHR$(13)AND IN$<>""THEN POKE 2
     04,1:PRINT"{SPACE,LEFT,SPACE}":IN=VAL
     (IN$):RETURN                          <093>
1150 IF EI$="-"THEN 190                    <228>
1160 IF LEN(IN$)=>15 THEN 110              <100>
1170 IF EI$="."THEN 190                    <249>
```

Listing 1. Die Routine zur Zahleneingabe (ab Zeile 1080) ist völlig durcheinandergeraten

```
1000 REM <><><><><><><><><><><><><><><><>    <084>
1010 REM <>                          <>    <016>
1020 REM <>     KNOBELN MIT BASIC    <>    <249>
1030 REM <>                          <>    <038>
1040 REM <> (C) BY DIRK NEUMEISTER   <>    <212>
1050 REM <>                          <>    <058>
1060 REM <><><><><><><><><><><><><><><><>    <146>
1070 END :                                 <009>
1080 NEXT                                  <074>
1090 FOR A=1 TO 49:IF ZAHL%(A)=1 THEN PRIN
     T A;                                  <136>
1100 FOR A=1 TO 6:REM TIPPVORSCHLAEGE EING
     EBEN                                  <097>
1110 FOR A=1 TO 6:IF ZAHL%(EIN(A))=1 THEN
     ZAELER=ZAELER+1                       <195>
1120 ZAHL%(ZUFALL)=1:NEXT                  <011>
1130 FOR A=1 TO 6:REM LOTTOZAHLEN ERMITTEL
     N                                     <213>
1140 REM LOTTOSIMULATION                   <170>
1150 NEXT:PRINT:PRINT:PRINT"SIE HATTEN";ZA
     ELER;"ZAHL(EN) RICHTIG"               <099>
1160 PRINT:PRINT"DIE AKTUELLEN LOTTOZAHLEN
      SIND"                                <251>
1170 INPUT"{2SPACE}]{2LEFT}";EIN(A):IF EIN
     (A)<1 OR EIN(A)>49 THEN PRINT"UNGUELT
     IG":GOTO 170                          <125>
1180 PRINT"{CLR}BITTE GEBEN SIE IHRE ZAHLE
     N EIN"                                <010>
1190 NEXT                                  <184>
1200 ZUFALL=INT(RND(1)*49)+1:IF ZAHL%(ZUFA
     LL)=1 THEN 130                        <162>
1210 DIM ZAHL%(49)                         <045>
```

Listing 2. Nur mit der richtigen Reihenfolge der Programmzeilen können Sie Lotto spielen

```
1000 REM <><><><><><><><><><><><><><><><>    <084>
1010 REM <>                          <>    <016>
1020 REM <>     KNOBELN MIT BASIC    <>    <249>
1030 REM <>                          <>    <038>
1040 REM <> (C) BY DIRK NEUMEISTER   <>    <212>
1050 REM <>                          <>    <058>
1060 REM <><><><><><><><><><><><><><><><>    <146>
1070 END :                                 <009>
1080 IF F<2 THEN PRINT"{DOWN}SIE KOENNEN R
     ICHTIG RECHNEN"                       <137>
1090 IF A$<>"N"THEN RUN                    <059>
1100 PRINT"SIE HABEN";R;"AUFGABE(N) GELOES
     T"                                    <020>
1110 INPUT"ANTWORT";A%:L%=A*B+C            <127>
1120 PRINT"{UP,13RIGHT}FALSCH{2SPACE}LOESU
     NG:";L%:F=F+1                         <229>
1130 IF A*B+C>100 THEN 120                 <159>
1140 TI$="000000"                          <103>
1150 PRINT"{2DOWN}NOCH EINMAL [J/N]"       <213>
1160 IF R<10 THEN PRINT"{DOWN}SIE MUESSEN
     SCHNELLER WERDEN!!"                   <199>
1170 IF A%=L%THEN PRINT"{UP,13RIGHT}RICHTI
     G":R=R+1:GOTO 190                     <252>
1180 IF F>1 THEN PRINT"{DOWN}SIE MACHEN ZU
      VIELE FEHLER!"                       <184>
1190 PRINT A"*"B"+"C"= ?                   <224>
1200 IF TI<3600 THEN 120:REM EINE MINUTE   <000>
1210 GET A$:IF A$=""THEN 260               <228>
1220 A=INT(RND(1)*10)+2:B=INT(RND(1)*10)+2
     :C=INT(RND(1)*10)+20                  <010>
1230 REM RECHENUEBUNGEN                    <020>
1240 IF R>9 THEN PRINT"{DOWN}SIE KOENNEN S
     CHNELL RECHNEN"                       <233>
```

Listing 3. Beweisen Sie, wie gut Sie Kopfrechnen können – vorher müssen Sie allerdings die Zeilen ordnen

Haben Sie die ursprüngliche Reihenfolge herausgefunden, sollten Sie anschließend eine Zeile ergänzen, bevor Sie das Programm starten:

```
10 GOSUB 100:END
```

Die Routine läßt sich sehr einfach in eigene Programme einbinden.

Listing 2:

Das korrigierte Programm ist eine Lottosimulation: Zunächst geben Sie sechs Zahlen ein, die Ihre Lottozahlen darstellen. Diese werden mit sechs Zufallszahlen verglichen. Nach der Auswertung erfahren Sie, wie viele Richtige Sie erzielt haben.

Listing 3:

Kopfrechnen ist angesagt mit dem dritten Programm. Eine Minute lang werden Ihnen über Zufallszahlen Rechenaufgaben gestellt. Das Programm läßt sich gut auf verschiedene Rechenarten ändern.

Wenn Ihnen diese Art zu knobeln gefällt und Sie Ideen für weitere Beispielprogramme haben, schreiben Sie uns.

(D. Neumeister/ef)

Der C 64 als Psychiater

Möchten Sie sich einmal mit Ihrem C64 unterhalten, ihm Fragen stellen oder auf dessen Äußerungen antworten? Alles kein Problem mit »Eliza«: Dieses Programm macht aus Ihrem Computer einen »intelligenten« Gesprächspartner.

In den Anfangszeiten der Forschung über Künstliche Intelligenz (KI) stellte »Eliza« (Listing 1) einen revolutionären Fortschritt dar. Obwohl das Programm bereits über 25 Jahre alt ist, stellt es ein vorzügliches Beispiel für KI dar. Entwickelt wurde es in den sechziger Jahren von dem Computerwissenschaftler Josef Weizenbaum. Ziel war es, einen Psychiater zu simulieren.

Die Simulation war so erfolgreich, daß in der Tat viele Gesprächspartner damals dachten, hinter dem Computer verberge sich ein Mensch. Verblüffend war auch der geringe Speicherplatzbedarf: »Eliza« benötigte nur 8 KByte RAM. Dennoch lief das Programm früher nur auf einem Großrechner. Kein Wunder, denn niemand dachte daran, daß es bereits 1983 Heimcomputer mit 64 KByte RAM geben würde.

Ende der siebziger Jahre verbreitete sich »Eliza« auch in Deutschland. Bis heute hat das Programm nichts an Aktualität und Wirkung eingebüßt. Wer sich heute mit Künstlicher Intelligenz oder mit der Programmierung intelligenter Adventures beschäftigt, kommt an »Eliza« nicht vorbei.

Wenn Sie Ihre »Probleme« von »Eliza« analysieren lassen wollen, benötigen Sie Englischkenntnisse oder zumindest ein Wörterbuch.

Um Ihnen einen ersten Eindruck darüber zu vermitteln, wie das Programm arbeitet, wollen wir Ihnen zunächst einen Ausschnitt aus einem Dialog vorstellen, wie er sich jederzeit mit »Eliza« entwickeln kann:

```
HI, I AM ELIZA TELL ME YOUR PROBLEM
> I HATE SCHOOL
SAY, DO YOU HAVE ANY PSYCHOLOGICAL PROBLEMS?
>I CAN'T GET NO SATISFACTION
HOW DO YOU KNOW YOU CAN'T GET NO SATISFACTION
>I JUST DON'T HAVE ANY
WHAT DOES THAT SUGGEST TO YOU?
>I NEED GIRLS AND DRUGS AND ROCKN ROLL
I SEE
>CAN YOU GIVE ME DRUGS?
DON'T YOU BELIEVE THAT I CAN GIVE  DRUGS!
>I LIKE YOU ELIZA
WE WERE DISCUSSING YOU -- NOT ME
>BY THE WAY - MY NAME IS MIKE
NAMES DON'T INTEREST ME
```

Wenn Sie sich eine Weile mit dem Eliza-Programm »unterhalten« haben, haben Sie sicherlich festgestellt, daß nicht selten relativ unsinnige Antworten zustandekommen. Daß im Programm nicht viel Intelligenz stecken kann, sieht man schon am Listing: Das Hauptprogramm besteht aus nur zirka 50 Zeilen, der restliche Teil aus DATA-Zeilen. In der Tat, Eliza hat keine Ahnung von dem Gesprächsinhalt der Dialoge, die sie führt. Der Eliza-Effekt, wie wir es einmal nennen wollen, besteht also darin, aus eingegebenen Sätzen und Antworten neue Fragen zu formulieren. Daß Eliza Antwort gibt, ohne nachzudenken und darauf verzichtet, die Antwort auf Sinngehalt zu untersuchen, zeigt sich dann, wenn man anstelle von einfachen Sätzen wirre Zeichenfolgen eingibt.

Kurzinfo: Eliza

Programmart: Beispiel für »Künstliche Intelligenz«
Laden: LOAD "ELIZA",8
Starten: Nach dem Laden RUN eingeben
Besonderheiten: Umsetzung für den C64 nach einem Programm von Josef Weizenbaum. Die Sprache ist Englisch.

Wir müssen an dieser Stelle zugeben, daß der Beispieldialog, den wir Ihnen vorhin gezeigt haben, nur deshalb relativ intelligent erscheint, weil wir durch gewählte Antworten (unter Kenntnis, wie Eliza arbeitet,) vermeiden konnten, daß Eliza unsinnige Antworten gibt.

Geben Sie nach Programmstart doch bitte einmal den folgenden Satz ein:

»CAN YOU X«

Eliza antwortet mit »DON'T YOU BELIEVE I CAN DO X«. Nach dem Schlüsselbegriff CAN YOU hat Eliza also einfach den Satzrest X abgeschnitten und an einen ihrer Standardsätze angehängt. Hätte man als ersten Satz

»CAN YOU DANCE« eingegeben, so hätte Eliza mit »DON'T YOU BELIEVE THAT I CAN DANCE« geantwortet. Geben wir jetzt nochmals »CAN YOU X« ein. Eliza antwortet jetzt mit:

»PERHAPS YOU WOULD LIKE TO BE ABLE TO X«. Beim dritten gleichen Versuch antwortet Eliza schließlich mit »YOU WANT ME TO BE ABLE TO X«. Gibt man zum viertenmal »CAN YOU X« ein, so erhält man wieder die erste Antwort. Eliza hat also nur eine begrenzte Anzahl von Antworten und Fragen zu bestimmten Schlüsselwörtern parat.

Eliza ist also eigentlich gar kein KI-Programm, sondern lediglich eine einfache Gesprächssimulation. Trotzdem wollen wir das Programm ausführlich besprechen, denn es läßt sich vieles aus ihm herausholen. Es bietet sich zum Beispiel sehr an, eine eliza-artige Routine in ein Text-Adventure einzubauen. Man könnte die Schlüsselwörter von Eliza speziell auf das Adventure abrechnen. Der Spie-

ler könnte dann einen richtigen Dialog mit dem Computer führen, der in etwa so aussehen könnte:

Spieler: Ich weiß hier nicht mehr weiter.
ADV: Denken Sie halt mal scharf nach!
Spieler: Ich krieg die verdammte Tür nicht auf!
ADV: Sie brauchen ein Hilfsmittel, wenn Sie die Tür öffnen wollen.
Spieler: Womit läßt sich die Tür öffnen?
ADV: Stellen Sie sich nicht so dumm an.
Spieler: Womit läßt sich die Tür öffnen?
ADV: Sie kommen bestimmt nicht weiter, wenn Sie immer wieder auf der gleichen Frage herumhacken.

Und so kann es noch lange weitergehen.

Dies sieht momentan natürlich wieder viel intelligenter aus, als es in Wirklichkeit ist. Auf jeden Fall bietet es sich an, das Eliza-Programm einmal genau unter die Lupe zu nehmen, um das Prinzip zu verstehen, das hinter dem Bilden der Antworten steht: Die wichtigsten Informationen werden der vorausgegangenen Frage entnommen. Alle Gesprächspartner-Simulationsprogramme bauen irgendwie auf dem Eliza-Prinzip auf. Es ist also unbedingt erforderlich, die Programmierung der Eliza zu verstehen, wenn man eigene Simulationsprogramme dieser Art schreiben will. Wie gesagt, das Eliza-Programm benötigt lediglich 8 KByte RAM. Das heißt, uns stehen noch 30 KByte Basic-RAM zur Verfügung. Nehmen wir alle 64 KByte RAM und bedienen uns auch noch einer relativen Datei auf Disk, so können wir Programme entwickeln, die den »IQ« der Eliza um ein Hundertfaches übersteigen.

Eliza unter der Lupe

Das Eliza-Listing läßt sich in zwei große Hauptabschnitte unterteilen:
dem Steuerprogramm (Zeile 5 bis 620) und den DATA-Zeilen (Zeile 1050 bis 2250).

Programmieranfänger sehen im Steuerprogramm lediglich ein Wirrwarr von Stringoperationen. Man muß schon ziemlich genau hinschauen, um zu erkennen, was sich in den einzelnen Zeilen des Steuerprogramms abspielt. Lassen Sie uns dies jetzt einmal gemeinsam tun:

Eliza: Dokumentation zum Listing

10 bis 30 Unterprogramm zum Positionieren des Zeigers auf die DATA-Zeilen. Um das Programm möglichst klein zu halten (es stammt schließlich von einem PET 2001 mit nur 8 KByte), war es nicht möglich, die gesamten DATAs in ein Stringfeld einzulesen. Deshalb wird dieses Unterprogramm benutzt, um auf einen bestimmten Satz N zu positionieren. Der DATA-Zeiger wird einfach auf das erste DATA gesetzt (mittels RESTORE), dann werden N-Elemente mittels READ überlesen. Es hat auch seinen guten Grund, daß dieses Unterprogramm am Kopf des Listings steht: Je niedriger die Zeile ist, in der ein Unterprogramm beginnt, um so schneller wird es mit GOSUB aktiviert. Da bei Eliza ständig positioniert werden muß, lohnt es sich aus Geschwindigkeitsgründen, das Unterprogramm voranzustellen.

100 bis 140 Hier wird eine Schlüsselbegriff-Tabelle erzeugt. In die Variablen S%, R%, N% (DIM-Felder) werden die Zahlendatas ab Zeile 2530 eingelesen – also alle Zahlendatas, die es überhaupt gibt. Diese Tabelle ist die wichtigste Tabelle im ganzen Programm (und auch die einzige).

Die gesamte Datei läßt sich in drei Bereiche gliedern:

Bereich 1 mit N1-Sätzen: enthält alle Schlüsselwörter, die Eliza kennt.
Bereich 2 mit N2-Sätzen: Hilfsverben
Bereich 3 mit N3-Sätzen: Kommentare

160 Beginn des Dialogs

170 »Befehlseingabe« in den String I$

200 bis 220 Alle ' werden aus I$ herausgekürzt.

Beispiel: Aus »I'M HUNGRY« wird »IM HUNGRY«. Dies ist notwendig, um später I AM und IM gleichsetzen zu können. Im Prinzip ist dieses Kürzen nicht unbedingt notwendig, da in der Schlüsselwort-Tabelle I AM und I'M aufgeführt werden.

230 In P$ ist jeweils die letzte Befehlseingabe gespeichert. Es wird hier dafür gesorgt, daß der Anwender nicht zu schnell das Eliza-Schema begreift, indem er mehrmals hintereinander die gleiche Antwort erhält. Dies gilt allerdings nur bei Verlegenheitsfällen (Erklärung folgt).

280 Der DATA-Zeiger wird auf den ersten Satz des ersten Dateibereiches positioniert.

300 bis 340 Der Befehlsstring I$ wird auf Schlüsselwörter hin durchsucht. Es wird zum Beispiel gesucht, ob »CAN YOU« im String vorkommt.

325 Wird ein Schlüsselwort gefunden, so speichert
S die Nummer des Schlüsselwortes (CAN YOU = 1, CAN I = 2 etc.),
T ab welchem Zeichen von I$ das Schlüsselwort steht.
F$: das Schlüsselwort selbst

365 Prüfen, ob ein Schlüsselwort gefunden wurde (S größer 0). Wurde eines gefunden, so speichert
K: die Nummer des Schlüsselwortes (K=S),
L: wo Schlüsselwort in I$ steht (L=T).
Sprung nach 100

370 Es wurde kein Schlüsselwort gefunden. In diesem Fall erhält K den Wert von N1 (=38). Man kann aber auch sagen, daß doch ein Satz gefunden wurde – Satz 38, der bei uns NOKEY FOUND heißt. Das heißt, immer wenn kein Schlüsselwort gefunden wird, positioniert Eliza auf Satz 38.

400 Ein Schlüsselwort wurde gefunden. N=N1

405 Jetzt wird auf den zweiten Bereich positioniert.

410 Nehmen wir einmal an, der Befehlssatz lautete »CAN YOU X«. In diesem Fall nimmt C$ den Wert »X« an.

420 bis 540 Änderung der Bezugswörter. Beispiel: Aus »YOU APPEAR IN MY DREAMS« als C$ wird nach der Umwandlung gemäß Bereich 2 »APPEAR IN YOUR DREAMS«.

560 bis 570 Positionieren auf dritten Datenblock.

580 Positionieren aus SATZ R%(K) des dritten Bereiches. Warum R%(K) und nicht einfach K? Ganz einfach – Eliza bedient sich eines Tricks, um bei gleichen Schlüsselwörtern nicht immer identische Antworten geben zu müssen. Betrachten Sie bitte die Dimensionierungsanweisung im Listing. Es gibt drei Feldvariablen: S%(X), R%(X) und N%(X). Nehmen wir einmal an, Eliza erhält viermal hintereinander das gleiche Schlüsselwort,
zum Beispiel »CAN YOU SWIM«
»CAN YOU DANCE« etc.
Immer lautet das erste Schlüsselwort »CAN YOU«. Das erste Mal antwortet Eliza mit dem Satz R%(1) des dritten Bereiches. Wäre das Schlüsselwort zum Beispiel ARE YOU, so würde sie mit R%(9) antworten.

590 Jetzt wird R%(1) um den Wert 1 erhöht. Überschreitet R%(1) den Wert N%(1), so wird R%(1) auf den Wert S%(1) zurückgesetzt. Das Listing zeigt uns, daß es zum Schlüsselwort CAN YOU drei mögliche Antwortsätze gibt, die in den Sätzen 1 bis 3 stehen. In S%(X) steht immer gespeichert, welcher Satz der erstmögliche Antwortsatz zu einem bestimmten Schlüsselwort ist. In N%(X) ist es der letzte. So wird erreicht, daß

nicht allzuschnell auffällt, wie dumm Eliza eigentlich ist, solange er keine Auswahl hat.

Nun haben wir auch schon das komplette Programm Eliza dokumentiert. Es kann sein, daß es für einen Anfänger schwierig ist, das Programmkonzept der Eliza komplett zu verstehen. Unsere Dokumentation kann Ihnen nur helfen; verstehen werden Sie das Programm jedoch erst dann, wenn Sie es eigenständig Schritt für Schritt durcharbeiten.

Im folgenden möchten wir das Eliza-Konzept noch einmal mit ganz einfachen Worten zusammenfassen:

Der Eliza-Effekt besteht darin, aus einer Antwort eine neue Frage zu formulieren.

Eliza untersucht eingegebene Sätze auf Schlüsselwörter hin. Nehmen wir einmal an, jemand gibt als Antwortsatz »I HATE SCHOOL« ein. Eliza macht sich nun keineswegs die Mühe, den ganzen Satz in einzelne Wörter zu zerlegen.

Eliza beruft sich vielmehr zunächst auf die Tabelle der Schlüsselwörter. Zunächst nimmt sie das erste Wort der Tabelle (CAN YOU) als Schlüsselwort beziehungsweise Vergleichsstring K$. Nun wird der gesamte Befehlsstring I$ durchsucht.

```
FORI=1 TO LEN(I$)
: IF MID$(IS,I,LEN(K$))=K$ THEN REM GEFUNDEN
NEXTI
```

Das Suchen erfolgt also einfach mittels Durchlaufen des gesamten Befehlsstrings und Vergleichen mit dem Schlüsselwort K$. Dieser Suchalgorithmus ist sehr einfach, dennoch reicht er hier aus.

Aus dem Listing können wir lesen, daß Eliza bei dem Befehlsstring »I HATE SCHOLL« mit Sicherheit kein Schlüsselwort finden wird. Daher wird Eliza zu »I HATE SCHOOL« auch keine geistreichen Bemerkungen geben können. Sie bringt eine Ausweichfrage – sie weicht also einfach auf ein anderes Thema aus (wie wir es auch tun, wenn wir nicht mehr weiter wissen). Nach einer erfolglosen Schlüsselwortsuche steht der Tabellenzeiger auf Satz 38 (NOKEYFOUND). Der letzte Satz der Schlüsselworttabelle ist somit immer der Hinweise auf die Ausweichfragen (z.B. SAY DO YOU HAVE ANY PSYCHOLOGICAL PROBLEMS) oder Verlegenheitsantworten (I SEE).

Aus dem Listing können wir erkennen, wo die Ausweichsätze stehen: S%(38) zeigt auf den Beginn und N%(38) auf das Ende beziehungsweise den letzten der Verlegenheitssätze. R%(38) ist, wie bereits dokumentiert, der Zeiger auf den Ausweichbereich. Er wird immer dann um 1 erhöht (auch zurückgestellt wenn größer als N%), wenn Eliza auf den Ausweichbereich zugegriffen hat. Eliza kennt genau sieben Ausweichsätze. Das heißt, Eliza wiederholt sich bei jeder siebten Ausweichantwort. Von der Größe der einzelnen Bereiche, also dem Antwortenreichtum eines Eliza-Programms, hängt es ab, wie schnell die eigentliche »Dummheit des Programms« vom Spieler erkannt wird.

Der Eliza-Effekt

Hat Eliza beim ersten Durchlauf also kein Schlüsselwort gefunden, so bricht sie alle weiteren »Berechnungen« ab und antwortet sofort mit einer Verlegenheitsantwort oder stellt eine Ausweichfrage. Dies kann man mit einem Adventure-Parser vergleichen, der abbricht, wenn er ein Wort nicht kennt – ICH KENNE DAS WORT X NICHT.

Lassen Sie uns jetzt jedoch einmal von einem interessanten Befehlssatz ausgehen. Unter Befehlssatz verstehen wir alle Arten von Sätzen, die ein Spieler beziehungsweise Anwender auf einen INPUT-Befehl hin eingibt. So zum Beispiel »YOU APPEAR IN MY DREAMS«. Eliza findet nun das Schlüsselwort »YOU« – Satz 13 in Bereich 2. Anhand des Listings können wir herausfinden, daß Eliza mit »We were discussing you – not me« antworten muß. Sicher haben Sie bereits bemerkt, daß im Listing manche Sätze mit einem Sternchen enden und andere wiederum nicht. Sätze, die nicht mit einem Sternchen enden, werden direkt ausgegeben, also so wie sie im Listing stehen. Der Antwortsatz »WE WERE DISCUSSING YOU – NOT ME« steht dort an Stelle 32 und hat am Ende kein Sternchen. Wie wir bereits wissen, steht der YOU-Zeiger jetzt auf Satz 33. Bei der nächsten Satzeingabe muß also als Antwort Satz 33 kommen – wenn im Befehlssatz wieder das Schlüsselwort YOU gefunden wird. Probieren wir dies doch einfach einmal aus. Nach Eingabe von »YOU APPEAR IN MY NIGHTMARES« folgt in der Tat die Antwort »OH, I APPEAR IN YOUR NIGHTMARES«. Der Satz lautet jedoch nur »OH, I« und ist mit einem Sternchen versehen. Bei der Antwort wurde zu Satz 33 also »APPEAR IN YOUR NIGHTMARES« zugefügt. Vergleichen wir nun einmal Eingabe und Ausgabe:

Eingabe: YOU APPEAR IN MY NIGHTMARES
Ausgabe: OH, I APPEAR IN YOUR NIGHTMARES

Hier sieht man ganz deutlich, was passiert ist. Eliza hat vom Eingabesatz den Satzbereich nach dem Schlüsselwort YOU abgetrennt und anschließend an den Ausgabesatz 33 OH, I angefügt. Dieser abgetrennte Bereich taucht im Programm als C$ auf. Bevor C$ jedoch an Satz 33 angefügt wurde, veränderte sich noch etwas: Gemäß Bereich 2 wurde das Wort MY durch YOUR ersetzt.

Damit ist der Eliza-Effekt auch schon beschrieben.

Mit dem jetzt erworbenen Wissen sollte es Ihnen nicht mehr allzu schwer fallen, das Eliza-Programm selbst unter die Lupe zu nehmen.

Die Eliza-Methoden sind zwar sehr simpel, zeigen bei einer Analyse des Programms jedoch ganz einleuchtend, wie man Probleme der Sprachverarbeitung, zum Beispiel Grundlagen der Stringprogrammierung oder der Zerlegungsverfahren für Zeichenketten, angehen kann.

Vielleicht versuchen Sie selbst einmal, ein deutsches KI-Programm ähnlich Eliza zu entwickeln. Schicken Sie es doch ein. (M. Nickles/ef)

```
5 GOTO 100                                   <189>
10 RESTORE                                   <060>
20 FOR I=1 TO N:READ S$:NEXT                 <175>
30 RETURN                                    <088>
100 DIM S%(38),R%(38),N%(38)                 <245>
110 N1=38:N2=12:N3=118                       <032>
120 N=N1+N2+N3:GOSUB 10                      <050>
130 FOR X=1 TO N1                            <028>
140 READ S%(X),L:R%(X)=S%(X):N%(X)=S%(X)+L
    -1:NEXT                                  <134>
160 PRINT"HI, I AM ELIZA TELL ME YOUR PROB
    LEM                                      <235>
170 INPUT I$:I$=" "+I$+" "                   <191>
200 FOR L=1 TO LEN(I$)                       <030>
210 IF MID$(I$,L,1)="'"THEN I$=LEFT$(I$,L-
    1)+RIGHT$(I$,LEN(I$)-L):GOTO 210         <118>
220 NEXT                                     <230>
230 IF I$=P$THEN PRINT"PLEASE DON'T REPEAT
     YOURSELF":GOTO 170                      <202>
280 RESTORE:S=0                              <180>
300 FOR K=1 TO N1                            <096>
305 IF S>0 THEN 340                          <050>
310 READ K$                                  <242>
320 FOR A=1 TO LEN(I$)-LEN(K$)               <151>
325 IF MID$(I$,A,LEN(K$))=K$THEN S=K:T=A:F
    $=K$                                     <015>
330 NEXT                                     <086>
340 A=0:NEXT                                 <218>
365 IF S>0 THEN K=S:L=T:GOTO 400             <227>
370 K=N1:GOTO 560                            <243>
400 N=N1                                     <106>
405 GOSUB 10                                 <089>
410 C$=" "+RIGHT$(I$,LEN(I$)-LEN(F$)-L+1)    <142>
420 FOR X=1 TO N2/2:READ S$,R$               <208>
440 FOR L=1 TO LEN(C$)                       <004>
450 IF L+LEN(S$)>LEN(C$)THEN 510             <096>
```

Listing 1. »Eliza«. Mit Checksummer eingeben

```
480 IF MID$(C$,L,LEN(S$))<>S$THEN 510                <002>
490 C$=LEFT$(C$,L-1)+R$+RIGHT$(C$,LEN(C$)-
    L-LEN(S$)+1):L=L+LEN(R$)                         <093>
500 GOTO 540                                         <014>
510 IF L+LEN(R$)>LEN(C$)THEN 540                     <040>
520 IF MID$(C$,L,LEN(R$))<>R$THEN 540                <217>
530 C$=LEFT$(C$,L-1)+S$+RIGHT$(C$,LEN(C$)-
    L-LEN(R$)+1)                                     <161>
540 NEXT:NEXT                                        <035>
555 IF MID$(C$,2,1)=" "THEN C$=RIGHT$(C $,
    LEN(C$)-1)                                       <241>
560 N=N1+N2                                          <015>
570 GOSUB 10                                         <000>
580 FOR X=1 TO R%(K):READ F$:NEXT X                  <116>
590 R%(K)=R%(K)+1:IF R%(K)>N%(K)THEN R%(K)
    =S%(K)                                           <114>
600 IF RIGHT$(F$,1)<>"*"THEN PRINT F$:P$=I
    $:GOTO 170                                       <000>
620 PRINT LEFT$(F$,LEN(F$)-1);C$:GOTO 170            <126>
1050 DATA"CAN YOU","CAN I","YOU ARE","YOUR
     E","I DONT","I FEEL","WHY DONT YOU"             <101>
1060 DATA"WHY CANT I","ARE YOU","I CANT","
     I AM","IM ","YOU ","I WANT","WHAT"              <036>
1070 DATA"HOW","WHO","WHERE","WHEN","WHY",
     "NAME","CAUSE","SORRY","DRAEM","HELLO
     "                                               <074>
1080 DATA"HI ","MAYBE"," NO","YOUR","ALWAY
     S","THINK","ALIKE","YES","FRIEND"               <034>
1090 DATA"COMPUTER","ELIZA","MONEY","NOKEY
     FOUND"                                          <209>
1230 DATA" ARE "," AM ","WERE ","WAS "," Y
     OU "," I ","YOUR ","MY "," IVE "                <208>
1235 DATA" YOUVE "," IM "," YOURE "                  <045>
1330 DATA"DON'T YOU BELIEVE THAT I CAN*"             <156>
1340 DATA"PERHAPS YOU WOULD LIKE TO BE ABL
     E TO*"                                          <249>
1350 DATA"YOU WANT ME TO BE ABLE TO*"                <113>
1360 DATA"PERHAPS YOU DON'T WANT TO*","DO
     YOU WANT TO BE ABLE TO*"                        <167>
1365 DATA"WHAT MAKES YOU THINK I AM*","DOE
     S IT PLEASE YOU TO BELIEVE I AM*"               <224>
1390 DATA"PERHAPS YOU WOULD LIKE TO BE*"             <065>
1400 DATA"DO YOU SOMETIMES WISH YOU WERE*"           <219>
1410 DATA"DON'T YOU REALLY*","WHY DON'T YO
     U*","DO YOU WISH TO BE ABLE TO*"                <059>
1440 DATA"DOES THAT TROUBLE YOU?","TELL ME
      MORE ABOUT SUCH FEELINGS."                     <157>
1460 DATA"DO YOU OFTEN FEEL*","DO YOU ENJO
     Y FEELING*"                                     <039>
1480 DATA"DO YOU REALLY BELIEVE I DON'T*",
     "PERHAPS IN GOOD TIME I WILL*"                  <137>
1500 DATA"DO YOU WANT ME TO*","DO YOU THIN
     K YOU SHOULD BE ABLE TO*"                       <002>
1520 DATA"WHY CAN'T YOU*","WHY ARE YOU INT
     ERESTED IN WHETHER OR NOT I AM*"                <159>
1540 DATA"WOULD YOU PREFER IF I WHERE NOT*
     ","PERHAPS IN YOUR FANTASIES I AM*"             <020>
1560 DATA"HOW DO YOU KNOW YOU CAN'T*","HAV
     E YOU TRIED?","PERHAPS YOU CAN NOW*"            <247>
1590 DATA"DID YOU COME TO ME BECAUSE YOU A
     RE*","HOW LONG HAVE YOU BEEN*"                  <255>
1610 DATA"DO YOU BELIEVE IT IS NORMAL TO B
     E*","DO YOU ENJOY BEING*"                       <229>
1630 DATA"WE WERE DISCUSSING YOU-- NOT ME.
     ","OH, I*"                                      <229>
1650 DATA"YOU'RE NOT REALLY TALKING ABOUT
     ME, ARE YOU?"                                   <015>
1660 DATA"WHAT WOULD IT MEAN TO YOU IF YOU
      GOT*","WHY DO YOU WANT*"                       <116>
1680 DATA"SUPPOSE YOU SOON GOT*","WHAT IF
     YOU NEVER GOT*"                                 <099>
1700 DATA"I SOMETIMES ALSO WANT*","WHY DO
     YOU ASK?"                                       <189>
1720 DATA"DOES THAT QUESTION INTEREST YOU?
     "                                               <086>
1730 DATA"WHAT ANSWER WOULD PLEASE YOU THE
      MOST?"                                         <076>
1740 DATA"WHAT DO YOU THINK?","ARE SUCH QU
     ESTIONS IN YOUR MIND OFTEN?"                    <078>
1760 DATA"WHAT IS THAT YOU REALLY WANT TO
     KNOW?","HAVE YOU ASKED ANYONE ELSE?"            <188>
1780 DATA"HAVE YOU ASKED SUCH QUESTIONS BE
     FORE?"                                          <023>
1790 DATA"WHAT ELSE COMES TO MIND WHEN YOU
      ASK THAT?"                                     <128>
1800 DATA"NAMES DON'T INTEREST ME.","I DON
     'T CARE ABOUT NAMES-- PLEASE GO ON."            <240>
1820 DATA"IS THAT THE REAL REASON?","DON'T
      ANY OTHER REASONS COME TO MIND?"               <239>
1840 DATA"DOES THAT REASON EXPLAIN ANYTHIN
     G ELSE"                                         <231>
1850 DATA"WHAT OTHER REASONS MIGHT THERE B
     E?","PLEASE DON'T APOLOGIZE!"                   <120>
1870 DATA"APOLOGIES ARE NOT NECESSARY."              <196>
1880 DATA"WHAT FEELINGS DO YOU HAVE WHEN Y
     OU APOLOGIZE?","DON'T BE SO DEFENSIVE
     !"                                              <001>
1900 DATA"WHAT DOES THAT DREAM SUGGEST YOU
     ?","DO YOU DREAM OFTEN?"                        <134>
1920 DATA"WHAT PERSONS APPEAR IN YOUR DREA
     MS?","ARE YOU DISTURBED BY YOUR DREAM
     S?"                                             <171>
1940 DATA"HOW DO YOU DO ... PLEASE STATE Y
     OUR PROBLEM."                                   <037>
1950 DATA"YOU DON'T SEEM QUITE CERTAIN.","
     WHY THE UNCERTAIN TONE?"                        <199>
1970 DATA"CAN'T YOU BE MORE POSITIVE?","YO
     U AREN'T SURE?","DON'T YOU KNOW?"               <223>
2000 DATA"WHY NO*","DON'T SAY NO IT'S ALWA
     YS SO NEGATIVE","WHY NOT?"                      <073>
2030 DATA"ARE YOU SURE?","WHY NO?","WHY AR
     E YOU CONCERNED ABOUT MY*"                      <214>
2060 DATA"WHAT ABOUT YOUR OWN*","CAN'T YOU
      THINK OF A SPECIFIC EXAMPLE?","WHEN?
     "                                               <137>
2090 DATA"WHAT ARE YOU THINKING OF?","REAL
     LY, ALWAYS?"                                    <076>
2110 DATA"DO YOU REALLY THINK SO?","BUT YO
     U ARE NOT SURE YOU*"                            <060>
2130 DATA"DO YOU DOUBT YOU*","IN WHAT WAY?
     ","WHAT RESEMBLANCE DO YOU SEE?"                <131>
2160 DATA"WHAT DOES THE SIMILARITY SUGGEST
      TO YOU?"                                       <241>
2170 DATA"WHAT OTHER CONNECTIONS DO YOU SE
     E?"                                             <160>
2180 DATA"COULD THERE REALLY BE SOME CONNE
     CTIONS?","HOW?"                                 <204>
2200 DATA"YOU SEEM QUITE POSITIVE.","ARE Y
     OU SURE?","I SEE.","I UNDERSTAND."              <074>
2240 DATA"WHY DO YOU BRING UP THE TOPIC OF
      FRIENDS?","DO YOUR FRIENDS WORRY YOU
     ?"                                              <038>
2260 DATA"DO YOUR FRIENDS PICK ON YOU?","A
     RE YOU SURE YOU HAVE ANY FRIENDS?"              <154>
2280 DATA"DO YOU IMPOSE ON YOUR FRIENDS?"            <004>
2290 DATA"PERHAPS YOUR LOVE FOR FRIENDS WO
     RRIES YOU.","DO COMPUTERS WORRY YOU?"           <242>
2310 DATA"ARE YOU TALKING ABOUT ME IN PART
     ICULAR?"                                        <057>
2320 DATA"ARE YOU FRIGHTENED BY MACHINES?"
     ,"WHY DO YOU MENTION COMPUTERS?"                <170>
2340 DATA"WHAT DO YOU THINK MACHINES HAVE
     TO DO WITH YOUR PROBLEM?"                       <063>
2350 DATA"DON'T YOU THINK COMPUTERS CAN HE
     LP PEOPLE?"                                     <214>
2360 DATA"WHAT IS IT ABOUT MACHINES THAT W
     ORRIES YOU?"                                    <087>
2370 DATA"SAY, DO YOU HAVE ANY PSYCHOLOGIC
     AL PROBLEMS?"                                   <080>
2380 DATA"WHAT DOES THAT SUGGEST TO YOU","
     I SEE","I'M NOT SURE I UNDERSTAND YOU
     ."                                              <129>
2410 DATA"COME COME ELUCIDATE YOUR THOUGHT
     S","CAN YOU ELABORATE ON THAT?"                 <239>
2430 DATA"THAT IS QUITE INTERESTING.","WHY
      DO YOU HAVE PROBLEMS WITH MONEY?"              <054>
2450 DATA"DO YOU THINK MONEY IS EVERYTHING
     ?"                                              <189>
2460 DATA"ARE YOU SURE THAT MONEY IS THE P
     ROBLEM?"                                        <235>
2470 DATA"I THINK WE WANT TO TALK ABOUT YO
     U, NOT{2SPACE}ABOUT ME","WHATS ABOUT
     ME?"                                            <151>
2490 DATA"WHY DO YOU ALWAYS BRING UP MY NA
     ME?"                                            <221>
2530 DATA 1,3,4,2,6,4,6,4,10,4,14,3,17,3,2
     0,2,22,3,25,3,28,4,28,4,32,3,35,5,40,
     9                                               <193>
2540 DATA 40,9,40,9,40,9,40,9,40,9,49,2,51
     ,4,55,4,59,4,63,1,63,1,64,5,69,5,74,2           <159>
2550 DATA 76,4,80,3,83,7,90,3,93,6,99,7,11
     6,3,113,3,106,7                                 <164>
```

Listing 1. Das Eliza-Programm (Schluß)

Peeks und Pokes

Das Salz in der Suppe für jeden Basic-Programmierer: eine Liste mit allen PEEKs und POKEs. Viele Programmier-Tricks werden durch diese Informationen erst möglich.

Viele Tricks - wie das Sperren der Taste <RUN/STOP>, den Cursor nicht mehr blinken lassen oder einen LIST-Schutz in das eigene Basic-Programm einbauen - sind nur mit Hilfe des POKE-Befehls möglich. Für Sie haben wir eine Liste mit allen wichtigen POKEs zusammengestellt.

Oft ist es sehr hilfreich, in einigen Speicherzellen mit dem PEEK-Befehl nachzuschauen, welcher Wert dort enthalten ist. Dazu finden Sie ebenfalls hilfreiche Informationen.

Die folgende Tabelle ist nach Speicherzellen geordnet. Dabei steht links die Nummer der Speicherzelle als Dezimalwert, wie es für die PEEK- und POKE-Befehle notwendig ist. Rechts daneben finden Sie den entsprechenden Hexadezimalwert (sinnvoll für alle, die sich auch für Maschinensprache interessieren). In der Beschreibung (rechts) stehen alle wichtigen Informationen zu den jeweiligen Speicherstellen.

Dezimal	Hex	Beschreibung
1	1	Inhalt 55 = normal Inhalt 54 = Basic ausgeschaltet (auf RAM umgestellt) Inhalt 53 = Basic und Kernel ausgeschaltet (auf RAM umgestellt).Es empfiehlt sich, Basic und Kernel vorher ins RAM zu POKEn, damit der Computer bei der Umschaltung nicht »abstürzt«.
10	a	Ist PEEK(10) = 0, so war der letzte Befehl LOAD. Ergibt PEEK(10) eine 1, so ist VERIFY eingegeben worden.
17	11	Mit diesem PEEK läßt sich abfragen, wie die letzte Variable zugewiesen wurde. Ist PEEK(17) = 0, dann war die letzte Variablenzuweisung ein INPUT, oder es hat noch keine Zuweisung stattgefunden. Ist PEEK(17) = 64, dann wurde die letzte Variable durch GET geholt. Bei PEEK(17) = 152 erfolgte die letzte Variablenübergabe durch einen READ-Befehl.
19	13	Nach POKE 19,64 wird beim nächsten INPUT-Befehl kein Fragezeichen ausgegeben. Allerdings kann man nachher durch Drücken der RETURN-Taste nicht mehr in die nächste Zeile gelangen. Es empfiehlt sich daher, nach dem INPUT-Befehl dies wieder mit POKE 19,0 rückgängig zu machen.
24	18	Mit POKE 24,0 wird FORMULAR TOO COMPLEX ERROR aufgehoben.
43/44	2b/2c	Die Anfangs-Speicherstelle des zur Zeit im Speicher befindlichen Basic-Programmes errechnet sich durch PEEK(43) + PEEK(44)*256.
45/46	2d/2e	Das Ende des Basic-Programmes erhält man durch PEEK(45) + PEEK(46)*256.
55/56	37/38	Das Ende des Basic-RAMs kann mit PEEK(55) + 256*PEEK(56) abgerufen werden.
57/58	39/3a	Die Zeilennummer, bei der nach einer Programmunterbrechung gestoppt wurde, errechnet sich durch PEEK(57) + 256*PEEK(58)
61/62	3d/3e	Zeiger auf Basic-Statement für CONT: Durch PEEK(61) + PEEK(62)*256 erhält man die Speicherstelle, die nach dem zuletzt ausgeführten Basic-Befehl liegt. Das heißt die Speicherstelle, von der sich der Basic-Interpreter bei CONT den nächsten Befehl holt. Tip: Bei CONT kommt öfter CAN'T CONTINUE ERROR vor, wenn man nach dem STOPen ein CLR eingegeben oder in irgendeiner Programmzeile etwas geändert hat. Liest man die Werte mit PEEK(61) und PEEK(62) nach der Unterbrechung aus, dann macht beispielsweise ein CLR nichts aus, wenn man vor CONT die zuvor ausgelesenen Werte wieder in diese Speicherstellen POKEt.
63/64	3f/40	Nummer der aktuellen DATA-Zeile: Mit PEEK(63) + PEEK(64)*256 erhält man die Nummer der DATA-Zeile, aus der gerade der letzte Wert geholt wurde. (Gut zum Finden von Fehlern in DATA-Zeilen geeignet.)
69/70	45/46	Zuletzt zugewiesene Variable: Bei normalen Fließkomma-Variablen ließt man den Wert aus mit PRINT CHR$(PEEK(69)) + CHR$(PEEK(70)) Bei Integer-Variablen (z.B XY%) erhält man den Namen durch PRINT CHR$(PEEK(69)-128) + CHR$(PEEK(70)-128) Strings (z.B. VX$) erhält man durch PRINT CHR$(PEEK(69)) + CHR$(PEEK(70)-128)
120	78	Nach Ausführung dieses POKEs nimmt der C64 keinerlei Befehle mehr an: POKE 120,2
144	90	Statusbyte: Mit PRINT ST läßt sich das Statusbyte abfragen.
145	91	Nach WAIT 145,16,16 wartet der C 64 auf den Feuerknopf von Joystick 1.
147	93	Wenn man die LOAD-Routine im Betriebssystem anspringt, holt es sich aus der Speicherzelle 147 die Information, ob LOAD oder VERIFY durchgeführt wird. Inhalt 0 = LOAD Inhalt 4 = VERIFY
157	9d	Ausgabe-Kontrolle: Inhalt 0 = Programm-Modus Inhalt 128 = Direktmodus Damit bei LOAD-Befehlen vom Programm aus die Mitteilungen SEARCHING, LOADING, oder VERIFYING auf dem Bildschirm erscheinen, setzt man vor dem LOAD-, VERIFY- oder SAVE-Befehl ein POKE157,128.
182	b6	PEEK(182) ruft die Zahl der Zeichenlesefehler ab.
183	b7	PEEK(183) gibt die Länge des Filenamens an.
184	b8	Die laufende Filenummer kann mit PEEK(184) abgerufen werden.
185	b9	Die aktuelle Sekundäradresse kann mit PEEK(185) angegeben werden.
186	ba	Die derzeitige Gerätenummer wird durch PEEK(186) abgerufen.
197	c5	Derzeitiger Tastendruck: PEEK(197)
198	c6	Nach WAIT 198,1 wartet der Computer auf eine gedrückte Taste.

Dezimal	Hex	Beschreibung
199	c7	RVS Flag: Eine reverse Zeichenausgabe bei PRINT erfolgt nach POKE 199,1. POKE 199,0 hebt den RVS-Modus auf.
200	c8	Zeiger auf Zeilenende. PEEK(200) gibt an, wieviele Zeichen die zuletzt eingegebene Zeile hatte.
201/202	c9/ca	PEEK(201) gibt die Zeile der aktuellen Cursorposition an, während PEEK(202) die Spalte abruft.
203	cb	Die derzeitig gedrückte Taste im CHR$-Modus kann mit PEEK(203) abgefragt werden. Ist keine Taste gedrückt, so steht der Wert 64 in diesem Byte.
204	cc	Nach POKE 204,0 bleibt der Cursor an, auch bei GET-Befehlen. Mit POKE 207,0:POKE 204,1 gelangt man wieder in den Normalzustand zurück.
211	d3	POKE 211,X - X bestimmt die Spalte der Cursorposition. Werte von 0 bis 39 können eingegeben werden. Siehe auch 214.
213	d5	Die Länge der momentanen Bildschirmzeile läßt sich über PEEK(213) abfragen.
214	d6	POKE 214,X - X bestimmt die Zeile der Cursorposition. Die Werte 0 bis 24 können eingegeben werden. Nach SYS 58640 erscheint der Cursor auf der entsprechenden Position.
215	d7	Das zuletzt eingegebene Zeichen im ASCII-Code liest man mit PEEK(215) aus
631-640	277-280	Tastaturpuffer (ASCII-Code): Über POKE-Befehle lassen sich beispielsweise Zeichen in den Tastaturpuffer eingeben.
641-644	281-284	Start- und Endadressen des Basic-RAMs: Durch Verändern der (vorgegebenen) Werte kann man die Größe des Basic-RAMs bestimmen, beispielsweise setzt POKE 643,0 : POKE 644,128 : SYS 64764 das Ende des Basic-RAM um 2 KByte nach oben.
646	286	POKE 646,X - Der Farbwert X bestimmt die Cursorfarbe (0-15).
647	287	POKE 647,X - X bestimmt die Farbe unter dem Cursor (Werte: 0-15).
649	289	Länge des Tastaturpuffers: Nach POKE 649,0 wird die gedrückte Taste nicht mehr auf dem Bildschirm ausgegeben. Die maximale Größe des Tastaturpuffers beträgt 10.
652	28c	Zähler für REPEAT-Verzögerung: Alle eingePOKEten Werte verzögern die Repeat-Funktion. Höherer Wert = größere Verzögerung.
653	28d	PEEK(653) zeigt, ob eine der drei Tasten <SHIFT>, <CBM> oder <CTRL> gedrückt wurde: Bit 0 = SHIFT-Taste, Bit 1 = CBM-Taste, Bit 2 = CTRL-Taste.
657	291	Nach POKE 657,128 ist eine Umschaltung zwischen Groß- und Kleinschrift von der Tastatur her unmöglich. POKE 657,0 erzeugt wieder den Normalzustand.
770	302	Durch einPOKEn eines beliebigen Wertes erfolgt die READY-Ausgabe unendlich oft. Nur Ausschalten hilft noch.
775	307	POKE 775,X - X aktiviert einen Listschutz. Nur über den Wert 167 kann der Listschutz aufgehoben werden.
776	308	Der Befehl POKE 776,1 zerstört das Programm.
777	309	Nach POKE 777,1 wird kein Befehl mehr ausgeführt. Der Cusor befindet sich in der linken Ecke.
781/782	30d/30e	Startadresse, ab der ein Programm geladen wird: Durch entsprechende POKE-Werte kann ein Basic-Programm in einen anderen Speicherbereich geladen werden.
784/785	310/311	USR-Vektor: Erfolgt der Einsprung in ein Maschinenprogramm über den USR-Befehl, so muß die Einsprung-Adresse vorher in diese beiden Byte gePOKEt werden.
788/789	314/315	IQR, Hardware-Interrupt: Das Betriebssystem springt ständig in diese Routine. Durch Ändern des Inhalts kann man eigene »interruptgesteuerte« Maschinen-Routinen ständig laufen lassen. POKE788,49 hebt die Wirkung der Stop-Taste auf. POKE 788,52 schaltet sie wieder ein.
792/793	318/319	RESTORE-Vektor: PEEK(792) + PEEK(793)*256 ergibt die Speicherstelle, an die beim Drücken der RESTORE-Taste gesprungen wird. Beispiel: Nach POKE 792,226 : POKE 793,252 wird beim Drücken der RESTORE-Taste ein Reset ausgelöst.
801/802	321/322	Durch die Kombination POKE 801,0:POKE 802,0: POKE 818,165 wird ein SAVE-Schutz aktiviert.
808	328	POKE 808,225 hebt die Wirkung der RUN/STOP-Taste auf, POKE 808,237 macht es rückgängig.
813	32d	Nach POKE 813,2 läßt sich ein Programm nicht mehr ändern.
819/819	332/333	Folgende Kombinationen bewirken ebenfalls einen SAVE-Schutz: 1. POKE 818,116 : POKE 819,196 2. POKE 818,34 : POKE 819,253
828/1019	33c/3fb	Kassettenpuffer: Nach LOAD oder VERIFY stehen im Kassettenpuffer folgende Informationen: 828: 1 = normales File, 3 = wurde mit SAVE "Name",1,1 abgespeichert. Solche Programme werden bei LOAD automatisch ab der Adresse geladen, von der sie abgespeichert wurden. 829/830: Hier ist die Startadresse des Programms abgelegt. 829 ist das Low-Byte, 830 das High-Byte. 831/832: Endadressen des Programms 833-1019: 186 Zeichen langer Programmname. Auf dem Bildschirm werden nur 16 Zeichen angezeigt, aber es lassen sich bis 186 Zeichen lange Pro-

Dezimal	Hex	Beschreibung
		grammnamen abspeichern. Der Kassettenpuffer ist auch gut zum Anlegen eigener Maschinenprogramme geeignet, sofern nicht mit Datasette sondern mit einer Diskettenstation gearbeitet wird.
36870	9006	PEEK(36870) liest die Horizontal-Position des Lichtgriffels.
36871	9007	PEEK(36871) liest die Vertikal-Position eines angeschlossenen Lichtgriffels.
42291		Koppeladressen angleichen: Falls Programme mit NEW gelöscht wurden, kann man mit SYS 42291 die Byte 2049 und 2050 wieder in Ordnung bringen, wenn vorher etwas anderes als 0 in diese Speicherzellen gePOKEt wird.
53270	d016	VIC-Control-Register: Ein horizontales Softscrolling kann gePOKEt werden. POKE 53270,0 = scrollen nach links POKE 53270,1 = scrollen nach rechts
53280	d020	POKE 53280,X - X bestimmt die Rahmenfarbe des Bildschirms.
53281	d021	POKE 53281,X - X legt die Hintergrundfarbe des Bildschirms fest.
56320	dc00	Joystick Port 2: WAIT 56320,16,16 wartet auf »Feuerknopf« WAIT 56320,4,4 wartet auf »Joystick nach links« WAIT 56320,1,1 wartet auf »Joystick nach oben« WAIT 56320,2,2 wartet auf »Joystick nach unten« WAIT 56320,8,8 wartet auf »Joystick nach rechts«
56321	dc01	Wie 56320, aber Joystick in Port 1.
56325	dc05	Die Blinksequenz des Cursors kann in diesem Byte bestimmt werden. Höherer Wert = langsamere Blinksequenz. Mit POKE 56325,58 wird die normale Blinksequenz festgelegt.
56464	dc90	Nach WAIT 56464,16,16 wartet der Computer auf den Feuerknopf des Joystick 2.
56576	dd00	Mit PEEK(56576) kann man die Pins PB0 bis PB7 vom User-Port (auf der Unterseite des Ports, siehe Handbuch) auslesen. Mit POKE in diese Speicherzelle kann man auch Ausgaben über den User-Port programmieren.
56578	dd02	Datenrichtungsregister für den User-Port: Jedes der Bits gibt die Datenrichtung für die Pins PB0 bis PB7. Dabei gilt: Bit 1 gesetzt (1) = Ausgang, Bit nicht gesetzt (0) = Eingang.
65409	ff81	SYS 65409 setzt den Video-Chip des C64 auf den Ursprungszustand zurück.
65493	ffd5	LOAD-Routine des Betriebssystems: Mit folgender kleiner Routine kann man Unterprogramme nachladen, ohne irgendwelche Basic-Pointer (wie beispielsweise die Zeiger auf die Endadresse, 45 und 46) zu verändern: POKE 186,1 : POKE 780,0 : POKE 781,0 : POKE 782,96 : POKE 183,0 : SYS 65493 Erklärung: 186,1 = Geräteadresse für Recorder (8 = Floppy) 781 und 782 gibt die Startadresse an, ab der das Programm geladen werden soll. 183,0 = kein Programmname. SYS 65493 = LOAD-Routine.
65511	ffe7	Durch SYS 65511 lassen sich alle Files schließen. So erspart man sich das lästige Eintippen von CLOSE1:CLOSE2:CLOSE3.... Dabei sollte aber beachtet werden, daß auf diese Weise nur der Kanal geschlossen wird, aber keine Dateien auf einer Disk.

(M. Kohlen/U. Jürgens/kn)

Master Mind als Vierzeiler

Es ist schon erstaunlich, in welcher Kurzform man dieses Spiel programmieren kann. Bei Master Mind geht es darum, eine Zahl, die sich der Computer »denkt«, zu erraten. Am Anfang gibt man die Stellenzahl der zu erratenden Zahl ein, sie darf maximal acht sein (man hat aber schon mit drei oder vier genug zu knobeln). In der ersten Spalte muß man nun jeweils eine Zahl eingeben, der Computer zeigt in den folgenden drei Spalten an:

1. Anzahl der richtigen Ziffern an der richtigen Stelle
2. Anzahl der richtigen Ziffern an der falschen Stelle
3. Anzahl der Versuche

Beispiel eines Spiels:

Stellen? 4

? 1123	1	0	1	? 8989	1	0	4
? 4456	0	0	2	? 7979	1	1	5
? 7789	2	1	3	? 7187	4	0	6

Es geht natürlich darum, die Zahl mit möglichst wenig Versuchen zu erraten. In Listing 1 müssen Sie beim Abtippen unbedingt alle Leerzeichen weglassen. In den Zeilen 2 und 3 sind für einige Befehle die Kurzzeichen zu verwenden, da die Zeilen sonst zu lang werden. Hier noch die Tabelle der Variablen:

S:	Anzahl der Stellen
E,E():	Eingabe, Ziffern der Eingabe
L():	Ziffern der Lösung
B(),C():	Belegungsvektoren für E bzw. L
R:	Richtige Ziffern an der richtigen Stelle
F:	Alle richtigen Ziffern
V:	Versuche
B,I,J:	Hilfs- und Laufvariablen

(Hans Haberl/kn)

```
1 INPUT"{CLR,DOWN}STELLEN";S:E=INT(10↑S*RN
  D(0)):GOSUB 2:FOR I=1 TO S:L(I)=E(I):NEX
  T:GOTO 3                                  <078>
2 FOR I=1 TO S:E(I)=E-10*INT(E/10):E=INT(E
  /10):C(I)=0:B(I)=0:R=R-(L(I)=E(I)):NEXT:
  RETURN                                    <227>
3 V=V+1:INPUT E:R=0:F=0:GOSUB 2:FOR I=1 TO
   S:FOR J=1 TO S:B=(L(J)=E(I))AND NOT B(I
  )AND NOT C(J)                             <129>
4 B(I)=B(I)+B:C(J)=C(J)+B:F=F-B:NEXT J,I:P
  RINT"{LIG.BLUE}"TAB(16)R"{3SPACE}"F-R"{3
  SPACE}"V:IF R<S THEN 3                    <100>
```

Listing 1. »Master Mind« bitte mit dem Checksummer (Seite 159) eingeben

Tips und Tricks für Basic-Programmierer

Haben Sie versehentlich nach längerem Programmieren NEW eingegeben? Wenn Sie mit dem Basic 2.0 des C64 arbeiten, ist Ihr Programm meist verloren. Es sei denn, Sie kennen einige Tricks, mit denen Sie es retten können. Wir haben Ihnen eine Vielzahl solcher Tips und Tricks zusammengestellt

Wer seine Programme ohne eine Basic-Erweiterung schreibt, vermißt häufig einige zusätzliche Befehle. Oft ist es gerade ein kleiner Trick oder ein kleines Hilfsprogramm, die das Programmieren erleichtern.

Unsere Mischung ist kunterbunt zusammengestellt: Sie finden beispielsweise Hilfen für den Umgang mit der Floppy ebenso wie für die Datasette.

Neben hilfreichen Tips gibt es kleine Anwendungsbeispiele, die Sie leicht in eigene Programme einbauen oder Ihren Bedürfnissen anpassen können. Alle Listings und Beispiele sind natürlich in Basic geschrieben.

1. »NEW« rückgängig machen

Wie schnell hat man einmal »NEW« eingetippt, um erst hinterher festzustellen, daß man das Programm noch gar nicht gespeichert hat. Mit dieser kleinen Routine können Basic-Programme nach einem »NEW« wieder vollständig zurückgeholt werden.

Man geht dabei so vor: Zuerst einmal muß das Listing 1 eingetippt und gestartet werden. Es erzeugt dann auf Diskette das Programm »AUTO-OLD«. Hat man nun einmal aus Versehen »NEW« eingegeben, so legt man die Diskette mit dem Old-Programm in die Floppy und gibt »LOAD"AUTO-OLD",8,1« ein. Das Old-Programm wird nun geladen und automatisch gestartet. Ein eventuell gelöschtes Basic-Programm ist wiederhergestellt.

(Georg Brandt/ef)

```
Ø :REM     AUTO-OLD BY G. BRANDT              <Ø94>
1Ø OPEN 1,8,1,"AUTO-OLD,P,W":FOR I=Ø TO 4Ø
   :READ A:PRINT#1,CHR$(A);:NEXT:CLOSE 1      <Ø59>
2Ø DATA 221,2,165,44,16Ø,1,145,43,32,51,16
   5,165,34,24,1Ø5,2,133,45,165,35,1Ø5        <Ø68>
3Ø DATA Ø,133,46,169,131,162,164,141,2,3,1
   42,3,3,1Ø8,2,3,139,227,221,2               <Ø79>
```

© 64'er

Listing 1. Die Auto-OLD-Routine können Sie mit Hilfe des Checksummers eingeben

2. Noch einmal: NEW rückgängig

Schnell ist es passiert, daß man einmal aus Versehen NEW eingetippt hat. Das Basic-Programm ist dann rettungslos verloren. Oder doch nicht? Nein, mit einigen POKEs und einem SYS-Befehl kann man es »wiederbeleben«:

```
POKE2050,1:SYS42291:POKE46,PEEK(35):POKE45,PEEK
(781)+2:CLR
```

In ganz seltenen Fällen kann es vorkommen, daß der Computer daraufhin einen »illegal quantity error« meldet. Dann muß man noch zusätzlich

```
POKE46,PEEK(35)+1:POKE45,PEEK(781)-254
```

eingeben.

Achtung: Das Ganze funktioniert nur, wenn man nach dem versehentlichen NEW-Befehl keine Variable definiert hat (zum Beispiel A=10)!

(Alexander Rühl/ef)

3. Professioneller Cursor

Kleines Listing mit großem Effekt: Listing 2 zeigt, wie man mit wenig Aufwand bei Texteingaben einen professionell aussehenden Cursor erzeugen kann. Einfach eintippen und ausprobieren!

(Jan C. Tchinitchian/ef

```
10 BL$="{LEFT}"                          <077>
20 PRINT"{CLR}"                           <008>
30 PRINT"BITTE GEBEN SIE TEXT EIN:"       <088>
40 PRINT BL$;                             <182>
50 GET X$:IF X$=""THEN 50                 <095>
60 IF X$=CHR$(13)THEN 80                  <202>
70 PRINT X$;:TE$=TE$+X$:GOTO 40           <216>
80 PRINT"SIE GABEN EIN:"                  <180>
90 PRINT TE$                              <058>
```

© 64'er

Listing 2. Professioneller Cursor

4. Turbo-Cursor

Wenn man seinen Cusor so weit beschleunigen möchte, daß man kein vernünftiges Wort mehr schreiben kann, genügen zwei POKEs:

```
POKE650,128:POKE56325,1
```

Der zweite POKE stellt die eigentliche Geschwindigkeitsbeeinflussung dar. Die Zahl 1 bedeutet größtmögliche Ge-

schwindigkeit (Werte zwischen 1 und 255). Der erste POKE dient nur dazu, die Tastatur auf Wiederholfunktion zu schalten. Versuchen Sie nun einmal, das Wort »COMPUTER« zu schreiben. Sie werden an Ihren Fähigkeiten zweifeln...

(Axel Stämmler/ef)

5. Hilfe beim Eintippen von DATA-Zeilen

Wenn Sie ein Programm abtippen, das sehr viele DATA-Zeilen enthält, schleichen sich fast automatisch irgendwelche Tippfehler ein. Meist werden diese DATA-Zeilen über eine FOR-NEXT-Schleife mit READ- und POKE-Anweisungen bearbeitet. Ein Tippfehler macht sich dann durch einen »illegal quantity error« bemerkbar. Nun kann man mit

```
LIST (PEEK(63)+PEEK(64)*256)
```

die fehlerhafte DATA-Zeile sofort auf den Bildschirm bringen. Voraussetzung ist natürlich, daß Sie sich nicht schon beim Eingeben der FOR-NEXT-Zeile vertippt haben.

(Oliver Twesten/ef)

6. DATA Eingabehilfe

Das kleine Programm ist eine Hilfe zum Eintippen von DATAs. In Zeile 1 wird nach der Startzeilennummer gefragt, ab der die DATAs beginnen und in welcher Schrittweite die Zeilennummer erhöht werden soll.

Nachdem das Programm mit RUN gestartet und die beiden Parameter eingegeben wurden, erscheinen 20 Zeilen mit der gewünschten Startzeilennummer und dahinter das Wort »DATA«.

Diese Zeilen können nun editiert werden.

Das Programm läßt sich beliebig verändern. Statt DATA kann man auch PRINT oder ähnliches einsetzen.

```
1 INPUT "STARTZEILENNUMMER";X: INPUT
"SCHRITTWEITE";Y: FOR Z=1 TO 20
2 PRINT X "DATA": X=X+Y: NEXT:PRINT
"{HOME}{CRSR DOWN}": END
```

(Frank Besler/ef)

7. Hilfe bei DATA-Wüsten

Kennen Sie das nicht auch? Man tippt eine meterlange DATA-Wüste in seinen Computer ein, startet sie und erfreut sich an Meldungen wie zum Beispiel: out of data error, illegal quantity error oder andere. Folge: heftiges Vergleichen der DATA-Ziffern.

Um dies zu verhindern, habe ich mir folgenden Trick einfallen lassen: Man nehme das zu bearbeitende Programm und gebe folgenden Einzeiler ein.

```
FOR I=1 TO 20000 : READA: PRINT A : PRINT : POKE 198,0 :
WAIT 198,1 : NEXT I
```

Nach jedem Tastendruck steht jeweils ein DATA-Wert auf dem Bildschirm, den Sie überprüfen sollten. Sollten Sie einen Fehler entdecken, sofort im Heft markieren und weiter überprüfen. Erst nachdem Sie einen OUT OF DATA ERROR erhalten, können Sie mit der Korrektur beginnen. Normalerweise sind die DATAs jetzt in Ordnung. Wenn nicht, haben Sie einen Fehler gemacht und müssen die Überprüfung wiederholen...

(Jens Ingo Jakubczyk/ef)

8. Tips zum Directory

Dieser Tip zeigt, wie man das Inhaltsverzeichnis einer Diskette teilweise laden kann.

Mit `LOAD"$:NAME",8` wird nur der Eintrag »NAME« geladen. Es werden Diskettenname, ID, Programmtyp, -länge und die freien Blocks ausgegeben.

Mit `LOAD"$:A*",8` werden alle Einträge, die mit »A« anfangen, geladen.

Mit `LOAD"$:???",8` werden alle Einträge, die drei Zeichen lang sind, von Diskette geladen.

(Sascha Sadewasser/ef)

9. Sein oder Nichtsein

Wie stellt man fest, ob eine Datei auf Diskette existiert oder nicht? Im Prinzip ganz einfach: Man versucht sie umzubenennen, und zwar mit dem Namen, den sie ohnehin schon besitzt. Dann kann es nur zwei Fehlermeldungen geben: »FILE EXISTS« oder »FILE NOT FOUND«. Im C 64-Basic sieht das so aus:

```
100 INPUT "PROGRAMMNAME"; N$
110 OPEN 15,8,15,"R:"+N$+"="+N$:INPUT#15,FM:
CLOSE15:IF FM=62 THEN PRINT"DATEI NICHT VORHANDEN":
GOTO 100
```

62 ist übrigens die Fehlernummer für »FILE NOT FOUND«.

(Ernst Kofler/ef)

10. Formatierte Strings

Wenn Sie mit einem PRINT-Befehl einen String auf dem Bildschirm ausgeben wollen, erleben Sie manchmal eine böse Überraschung: Liegt die Länge des Strings zwischen 40 und 255 Zeichen, wird nach dem 40. Zeichen der String abgeschnitten und in der nächsten Zeile fortgesetzt. Es spielt dabei keine Rolle, ob ein Wort auseinandergerissen wird. »Word-Wrapping«, wie es in jeder guten Textverarbeitung zu finden ist, kennt das Betriebssystem nicht.

Mit Listing 3 stellen wir Ihnen eine kleine Basic-Routine vor, die bei der Ausgabe eines Strings prüft, ob ein Wort beim Wechsel in die nächste Zeile auseinandergerissen wird. Paßt das Wort nicht mehr vollständig in die erste Zeile, wird es komplett in die nächste Zeile verschoben.

Nach dem Programmstart werden Sie gefragt, in welchem Bereich ein String ausgegeben werden soll. Zwei Tabulatoren stehen zur Verfügung, die den Bereich links und rechts abgrenzen. Zusätzlich können Sie den Zeilenabstand für die Ausgabe festlegen.

Die Routine läßt sich, wenn Sie die Zeilen, die nur der Demonstration dienen, entfernen, leicht in eigene Programme einbinden.

(A.Stranzinger/ef)

```
1 REM ***** FORMATIERTER STRING *****
    **   MADE    BY A.STRANZINGER **        <231>
2 A$="FORMATIERTE AUSGABE EINES STRINGS -
  IDEE VON A.STRANZINGER AM 01.10.88 "      <188>
3 AD=58732:L=LEN(A$)                        <072>
4 INPUT"{CLR,DOWN,SPACE}SPALTENBEGINN{3SPA
  CE}3{3LEFT}";SB                           <155>
5 INPUT"{DOWN,SPACE}SPALTENENDE{5SPACE}36{
  4LEFT}";SE                                <053>
```

Listing 3. Word-Wrapping für den C64

```
6 INPUT"{DOWN,SPACE}ZEILENABSTAND{3SPACE}3
  {3LEFT}";ZA                                <245>
7 PRINT"{CLR,WHITE}0---------1---------2--
  -------3---------":POKE 211,SB:SYS AD      <116>
8 FOR A=1 TO L:B$=MID$(A$,A,1)               <039>
9 IF B$=CHR$(32)THEN GOSUB 12:B$=MID$(A$,A
  ,1)                                        <030>
10 PRINT B$;:NEXT                            <250>
11 A=0:GOSUB 12:GOTO 8                       <119>
12 X=PEEK(211):Y=PEEK(214):P=SE-X            <108>
13 FOR B=A+1 TO L:C$=MID$(A$,B,1)            <211>
14 : IF C$=CHR$(32)THEN 16                   <101>
15 NEXT                                      <025>
16 C=B-(A+2):IF C<P THEN POKE 211,X:POKE 2
  14,Y:SYS AD:RETURN                         <237>
17 IF Y<20 THEN 20                           <251>
18 POKE 211,0:POKE 214,23:SYS AD             <162>
19 PRINT"{RVSON}DRUECKEN SIE F7 FUER EIN A
  NDERES FORMAT{SPACE,HOME}":POKE 198,0:W
  AIT 198,1:RUN                              <039>
20 POKE 211,SB:POKE 214,Y+ZA:SYS AD:A=A+1:
  RETURN                                     <085>
```

Listing 3. (Schluß)

11. Hilfe bei »file not found«

Manchmal starrt man als stolzer Besitzer einer Floppy-Station auf den Bildschirm und die Fehlermeldung »file not found«. Besonders schlimm ist dies, wenn der Fehler in einem Programm auftrat, man also nicht weiß, welches File denn nun nicht gefunden wurde. Kein Problem: Durch die Eingabe von »SYS 63123« (leicht zu merken) bringt der C 64 die Meldung »SAVING...« und den zuletzt verwendeten File-Namen auf den Bildschirm. Sie können das leicht testen: Tippen Sie »LOAD"XYZ",8« ein (plus RETURN-Taste) und anschließend »SYS 63123«. (Thomas Röder/ef)

12. Schreibschutz für Disketten

Ist es Ihnen nicht auch schon einmal passiert: Sie haben in mühevoller Kleinarbeit eine Diskette mit allen wichtigen Programmen zusammengestellt, und wie es der Zufall will, plötzlich befinden sich durch falsche Eingabe Files auf der Diskette, die nicht dort hingehören oder die Diskette ist neu formatiert. Gut, Files lassen sich löschen. Mit einer neu formatierten Diskette sieht das schon anders aus. Die auf ihr gespeicherten Programme sind verloren.

Der mechanische Schreibschutz (Aufkleber) ist auch nicht der Weisheit letzter Schluß, um solchen Zufällen zu begegnen. Wenn häufiger Daten auf der Diskette geändert werden sollen, muß der Schreibschutz jedesmal entfernt und nach erfolgreicher Änderung wieder angebracht werden. Um diesen mechanischen Schreibschutz zu umgehen, kann eine Diskette auch softwaremäßig geschützt werden (Listing 4).
Wie das funktioniert, soll kurz erklärt werden.

Auf der Diskette ist im 2. Byte auf Spur 18 Sektor 0 das Formatkennzeichen der 1541 gespeichert, und zwar die Zahl $41. Wird in dieses Byte eine andere Zahl geschrieben (mit einem Disketten-Monitor), so funktioniert das Beschreiben einer Diskette nur so lange, bis die Floppy-Station neu initialisiert wird (zum Beispiel aus- und wieder einschalten). Wird jetzt versucht, ein Programm oder Daten zu speichern, meldet die Floppy-Station einen Formatfehler, und nichts geht mehr. Dieser Software-Schutz funktioniert auch dann, wenn versucht wird, die Diskette mit

```
OPEN 15,8,15,"N:TEST"
```

zu formatieren. Das heißt, formatiert wird die Diskette mit dieser Befehlssequenz eigentlich nicht. Es wird nur das Inhaltsverzeichnis und ein Diskettenbereich (BAM) gelöscht, der Informationen darüber enthält, welche Diskettenblöcke belegt sind und welche nicht. Dieser Vorgang nennt sich auch »weich formatieren«.

Soll eine softwaremäßig geschützte Diskette »entschärft« werden, taucht ein Problem auf. Es läßt sich ja absolut nichts mehr speichern beziehungsweise ändern. Man muß also das Betriebssystem der Floppy-Station davon überzeugen, daß das richtige Formatkennzeichen an der richtigen Stelle steht. Dazu existiert im Speicher der Floppy-Station eine Speicherzelle, in die nach jeder Initialisierung das aktuelle Formatkennzeichen gespeichert wird. Es handelt sich um die Speicherzelle $101. Wird in diese Speicherzelle mit dem Befehl

```
PRINT#15,"M-W" CHR$(1)CHR$(1)CHR$(1)CHR$(65)
```

(Zeile 100) wieder das 1541 Formatkennzeichen ($41 beziehungsweise 65 oder A) geschrieben, läßt sich die Diskette wieder ganz normal beschreiben. Unter anderem kann nun auch wieder das Formatkennzeichen auf der Diskette geändert werden. (Bruno Henze/ef)

```
10 OPEN 15,8,15,"I":OPEN 2,8,2,"#"                <076>
20 PRINT#15,"U1 2 0 18 0"                         <248>
30 PRINT#15,"B-P 2 2":GET#2,FO$                   <113>
50 PRINT"{CLR,2DOWN,3SPACE}ALTES FORMATZEI
   CHEN = ";FO$                                   <189>
55 IF FO$="A"THEN X$="X"                          <031>
56 IF FO$<>"A"THEN X$="A"                         <136>
60 PRINT"{DOWN,3SPACE}NEUES FORMATZEICHEN
   = ";X$;"{LEFT}";                               <146>
70 POKE 198,0:WAIT 198,1                          <042>
80 A$=X$:GET X$:IF X$=CHR$(13)THEN X$=A$:G
   OTO 90                                         <043>
85 PRINT X$                                       <191>
90 IF FO$="A"THEN 200                             <172>
100 PRINT#15,"M-W"CHR$(1)CHR$(1)CHR$(1)CHR
    $(65)                                         <189>
200 PRINT#15,"B-P 2 2":PRINT#2,X$;                <071>
220 PRINT#15,"U2 2 0 18 0":PRINT                  <119>
230 IF X$="A"THEN PRINT"{2DOWN,3SPACE,RVSO
    N}SCHREIBSCHUTZ ENTFERNT"                     <197>
240 IF X$<>"A"THEN PRINT"{2DOWN,3SPACE,RVS
    ON}SCHREIBSCHUTZ AKTIVIERT"                   <229>
250 PRINT#15,"I":CLOSE 2:CLOSE 15                 <047>
```


Listing 4. Schreibschutz für Disketten

13. Programme nachladen

Wenn man von einem Basic-Programm aus ein zweites Programm (meist eine Maschinenroutine) nachladen möchte, ist dies gar nicht einmal so schwierig. Man muß nur ein paar Punkte beachten.

Der wichtigste wäre, daß der C 64 nach Beenden des Ladevorgangs einen »GOTO erste Programmzeile« ausführt. Wenn Sie zu Beginn eines Basic-Programms also eine Maschinenroutine (zum Beispiel Listing 2) durch »10 LOAD "XYZ",8« nachladen möchten, würde sich der C 64 in einer Endlosschleife »aufhängen«. Wir müssen uns beim ersten Durchlauf der Zeile 10 merken, daß das Programm jetzt schon geladen ist. Am einfachsten geschieht dies durch »10 IF A=0 THEN A=1 : LOAD "XYZ",8«. Beim ersten Programmstart mit »RUN« werden alle Variablen, also auch A, auf Null gesetzt. Die IF-Bedingung ist daher erfüllt; der LOAD-Befehl wird ausgeführt. Danach beginnt der C 64

wieder bei Zeile 10, die Variable A hat jetzt jedoch den Wert 1 - der LOAD-Befehl wird übersprungen und im Programm fortgefahren. Möchte man mehrere Programme nachladen, geschieht dies auf dieselbe Weise:

```
10  IF A=0 THEN A=1 : LOAD "P1",8
20  IF A=1 THEN A=2 : LOAD "P2",8
30  IF A=2 THEN A=3 : LOAD "P3",8
```

und so weiter.

Dies alles gilt jedoch nur für Maschinenprogramme! Möchte man ein Basic-Programm nachladen, so ist dies schon etwas komplizierter. Solange das nachgeladene Programm kürzer ist als das erste, genügt es, einen einfachen LOAD-Befehl einzusetzen (die IF-Abfrage kann entfallen, da das zweite Programm ja gleich gestartet wird). Wenn das nachgeladene Programm jedoch länger ist, werden sämtliche Variablen überschrieben.

Die allgemein günstigste Lösung soll hier kurz vorgestellt werden. Sie kann sowohl für Maschinen- als auch für Basic-Programme Verwendung finden.

Wenn Ihnen der Begriff »Tastaturpuffer« geläufig ist, können Sie diesen Absatz überspringen. Wenn nicht, geben Sie einmal folgende Zeile ein (ohne Zeilennummer), drücken <RETURN> und dann ein paar Tasten (bevor das »READY.« erscheint).

```
FOR I=1 TO 2000 : NEXT
```

Sie sehen, daß sich der C 64 Ihre Tastendrücke (bis zu zehn Stück) gemerkt hat. Er speichert sie in seinem »Tastaturpuffer«. Das Gute daran ist, daß wir durch ein paar gezielte POKE-Anweisungen in einem Basic-Programm diese Tastendrücke simulieren können. Das ist auch das Prinzip unserer Laderoutine:

Wir schreiben den LOAD-Befehl, der unser zweites Programm nachladen soll, auf den Bildschirm und machen dem C 64 vor, daß wir die RETURN-Taste gedrückt hätten. Dadurch wird der LOAD-Befehl dann ausgeführt. Also:

```
10  PRINT "{CLR,2DOWN}LOAD"CHR$(34)"XYZ"CHR$(34)",8
20  PRINT "{4DOWN} RUN{HOME}";
```

(Zur Erinnerung: Buchstaben innerhalb geschweifter Klammern dürfen Sie nicht ausschreiben, sondern müssen die entsprechenden Tasten drücken.)

Was bedeuten die »CHR$(34)« in der Zeile 10? Wie Sie vielleicht wissen, ist es nicht möglich, ein Anführungszeichen innerhalb von Anführungszeichen zu schreiben (mit »PRINT """«?). Dieses benötigen wir aber für den LOAD-Befehl. Das CHR$(34) bewirkt nun, daß an der entsprechenden Bildschirmposition ein Anführungszeichen ausgegeben wird.

Wenn Sie das Programm mit RUN starten, erscheint in der dritten Zeile der LOAD-Befehl und etwas weiter unten ein »RUN«. Wenn Sie jetzt zweimal <RETURN> drücken würden und ein Programm mit dem Namen »XYZ« auf Diskette hätten, würde dieses geladen und gestartet. Da wir das Ganze aber programmgesteuert machen wollen, müssen wir die beiden RETURNs von Basic aus simulieren, also in den Tastaturpuffer schreiben. Dies geschieht über:

```
30 POKE 631,13 : POKE 632,13
```

Der Tastaturpuffer hat nämlich die Adressen 631 bis 640. »13« ist der ASCII-Code der RETURN-Taste (siehe auch entsprechenden Anhang im Commodore-Handbuch zu Ihrem C 64). Jetzt müssen wir dem C 64 nur noch mitteilen, daß im Tastaturpuffer noch zwei unbearbeitete Tastendrücke vorliegen: `40 POKE 198,2`

Und damit die Tastendrücke auch ausgeführt werden, muß noch eine END-Anweisung folgen: `50 END`

Natürlich können Sie über diesen Trick nach dem Ladevorgang auch erst eine Variable definieren oder einen POKE ausführen. Dazu müßten Sie einfach die PRINT-Anweisung in Zeile 20 ändern.

Der Tastaturpuffer bietet eine Fülle von Möglichkeiten. Wer sich erst einmal mit ihm angefreundet hat, wird ihn nicht mehr missen wollen. Beim Ausprobieren von Programmen, die mit dem Tastaturpuffer arbeiten, ist es ratsam, den Tastenzähler bei Adresse 198 so lange auf Null stehenzulassen, bis auf dem Bildschirm die Positionierung der auszuführenden Befehle stimmt. Zu beachten wäre noch, daß Sie in den Tastaturpuffer nur ASCII-Werte POKEn dürfen (über PRINT ASC("Taste") oder Anhang im Handbuch herauszufinden). (Thomas Röder/ef)

14. Stichwort: Directory

Eine wirklich einfache Methode, das Disketten-Inhaltsverzeichnis ohne Programmverlust zu betrachten, ist folgende:

```
POKE254, PEEK(45): POKE255, PEEK(46): POKE44,192: LOAD "$",8
```

Das Directory kann nun gelistet werden. Alle anderen Eingaben wie »RUN«, »NEW« oder ähnliches führen jedoch zu einem »syntax error«. Zum Weiterarbeiten mit dem Basic-Programm sind noch drei POKEs einzugeben:

```
POKE45, PEEK(254): POKE46, PEEK(255): POKE44,8
```

Ein eventuell vorhandenes Basic-Programm bleibt erhalten und kann normal fortgesetzt werden. (Thomas Röder/ef)

15. Hilfe für Datasettenbesitzer

Kürzere Maschinenprogramme werden vom Programmierer gerne im Kassettenpuffer abgelegt. Den Floppy-Besitzer stört dies nicht. Er kann Programme laden und das Maschinenprogramm bleibt ihm erhalten. Nicht so beim Benutzer einer Datasette. Lädt er ein Programm von Kassette in den Bereich ab 828 ($033C), so wird es zerstört. Es gibt aber einen Trick, den Kassettenpuffer an einen anderen Speicherbereich zu legen. In den Speicherstellen 178 und 179 steht der Beginn des Kassettenpuffers. Normalerweise beginnt dieser bei 828. Durch Verändern dieses Vektors können wir den Kassettenpuffer beim Laden von Programmen schonen. Um ihn ans Ende des Speicherbereichs zu legen, schreiben Sie: POKE 179, PEEK (56)-2. (Herbert Kunz/ef)

16. Maschinenprogramme kopieren

Besitzt man kein Kopierprogramm, kann man Maschinenprogramme mit Hilfe des MSE kopieren: MSE laden - MSE starten - Programm laden - andere Diskette/Kassette einlegen - Programm mit CTRL-S wieder speichern. Basic-Programme lassen sich mit dieser Methode übrigens genauso kopieren. (Thomas Röder/ef)

17. Programme von Datasette nachladen

Mit Hilfe des folgenden POKEs wird das nächste Programm von der Datasette nachgeladen und automatisch gestartet:

```
POKE 631,131 : POKE 198,1 : END
```

Funktionsweise:

Dem C 64 wird durch den ersten POKE vorgetäuscht, es würde gerade die SHIFT-RUN/STOP-Taste gedrückt. Der Code dieser Tastenkombination (131) wird in den Tastatur-

puffer ab Adresse 631 geschrieben. Daraufhin muß dem Computer noch mitgeteilt werden, daß sich in diesem Tastaturpuffer ein noch nicht bearbeiteter Tastencode befindet (POKE 198,1). (Hans Ippisch/ef)

18. Adressenverwaltung

Trotz der geringen Länge des Programms bietet Listing 5 eine recht komfortable Adressenverwaltung.

Vier Menü-Punkte stehen nach dem Start zur Auswahl:

Liste: Gibt alle vorhandenen Datensätze auf dem Bildschirm aus.

Suchen: Prüft alle sieben zur Verfügung stehenden Datenfelder auf Übereinstimmung mit dem Suchbegriff.

Eingabe: Mit diesem Punkt werden neue Daten in die Datei übernommen.

Ende: Wurden neue Eingaben gemacht, wird die neue Datei unter dem Namen »ADRESSEN« auf Diskette gespeichert.

Alle neu eingegebenen Daten werden in Form von DATA-Zeilen an das Programm angehängt. Geben Sie viele Adressen ein, wird das Programm wesentlich länger. Die Zeile 100 enthält einen Zähler, wieviel Datensätze existieren: Die Zahl Null zeigt an, daß keine Daten vorhanden sind.

Eine auf Diskette vorhandene Datei wird zuerst gelöscht, bevor eine erweiterte gespeichert wird. Der alte Fehler des Überschreibens unter Verwendung des »Klammeraffen« wird umgangen.

Bei vielen Adressen ist die Lösung mit DATA-Zeilen nicht unbedingt zu empfehlen, die Verwendung einer sequentiellen Datenspeicherung hat dann Vorteile. Mit einer geringen Anzahl von Adressen ist diese Adressenverwaltung ein hervorragendes Beispiel dafür, daß gute Programme nicht immer sehr lang sein müssen. (G.Schrotz/et)

```
1 PRINT CHR$(14):D$=CHR$(17)+CHR$(17):I$=C
  HR$(18):C$=CHR$(147):H$=CHR$(19)           <193>
2 U$=" ADRESSEN ":A$(0)="1 LISTE":A$(1)="2
   SUCHEN":A$(2)="3 EINGABE"                 <166>
3 A$(3)="4 ENDE":B$(0)="ANREDE:":B$(1)="VO
  RNAME:":B$(2)="NAME:"                      <197>
4 B$(3)="STRASSE:":B$(4)="PLZ:":B$(5)="ORT
  :":B$(6)="TELEFON:":E$=CHR$(34)            <191>
5 PRINT C$;:PRINT TAB(2);I$;U$;D$;D$:FOR I
  =0 TO 3:PRINT TAB(2);A$(I);D$:NEXT I       <017>
6 PRINT TAB(2);D$;I$;" AUSWAHL 1-4 ";:INPU
  T X:IF X<1 OR X>4 GOTO 5                   <005>
7 ON X GOTO 8,10,14,17                       <243>
8 RESTORE:READ E:FOR I=1 TO E:FOR J=0 TO 6
  :READ E$(J):NEXT J:GOSUB 19:PRINT H$:FOR
   J=0 TO 6                                  <021>
9 PRINT TAB(13);D$;E$(J):NEXT J:POKE 198,0
  :WAIT 198,1:NEXT I:GOTO 5                  <189>
10 PRINT C$;D$;D$:INPUT"SUCHBEGRIFF: ";S$:
   RESTORE:READ E:FOR I=1 TO E:FOR J=0 TO
   6                                         <249>
11 READ E$(J):NEXT J:FOR J=0 TO 6:IF S$<>E
   $(J)GOTO 13                               <022>
12 GOSUB 19:PRINT H$:FOR K=0 TO 6:PRINT TA
   B(13);D$;E$(K):NEXT K:POKE 198,0:WAIT 1
   98,1                                      <086>
13 NEXT J,I:GOTO 5                           <219>
14 GOSUB 19:PRINT H$:FOR I=0 TO 6:PRINT TA
   B(13);D$;:INPUT E$(I):NEXT I:RESTORE:RE
   AD E                                      <073>
15 E=E+1:PRINT C$;"100DATA"E:FOR I=0 TO 6:
   PRINT 100+E*10+I;"DATA"+E$+E$(I)+E$:NEX
   T I                                       <232>
16 PRINT"GOTO1":POKE 631,19:FOR I=1 TO 9:P
   OKE 631+I,13:NEXT I:POKE 198,10:POKE 2,
```

Listing 5. Eine menügesteuerte Adressenverwaltung

```
   1:END                                     <184>
17 IF PEEK(2)=0 THEN END                     <065>
18 OPEN 15,8,15,"S:ADRESSEN":CLOSE 15:SAVE
   "ADRESSEN",8:END                          <182>
19 PRINT C$:FOR L=0 TO 6:PRINT TAB(2);D$;B
   $(L):NEXT L:RETURN                        <255>
100 DATA 0                                   <243>
```

Listing 5. (Schluß)

19. INPUT ohne Fragezeichen

Eine von vielen Möglichkeiten, das lästige Fragezeichen beim INPUT-Befehl wegzubekommen, ist die folgende:

`OPEN 1,0 : INPUT #1,A$ : CLOSE 1` (Thomas Röder/ef)

20. Rundungsfehler

Wie jedermann weiß, ist die Rechengenauigkeit des C 64 ziemlich gering. Manchmal werden gerundete Ergebnisse angezeigt, die zwar richtig aussehen, deren mit Hilfe der INT-Funktion ermittelter ganzzahliger Teil ganz anders aussieht, nämlich um 1 kleiner ist. Ein Beispiel:

1/50*100=2, aber

INT(1/50*100)=1!

Dies kann man verhindern, indem man die Zahl erst mit »STR$« in einen String umwandelt, mit »VAL« daraus wieder eine Zahl macht und dann die ganze Zahl ermittelt, also INT(VAL(STR$(1/50*100)))=2.

Dieser Trick ist vor allem nützlich bei Rechnungen, bei denen die 9/10-Rundung schon zu ungenau ist. (Wolfgang Müller/ef)

21. Hauptstadt-Quiz

Kennen Sie sich in Geographie aus? Listing 6 ist ein kleines Erdkunde-Quiz: Sie werden nach der Hauptstadt eines Landes gefragt. Geben Sie eine richtige Antwort, erscheint nach einer Bestätigung die nächste Frage. Nach einer falschen Antwort gibt das Programm zuerst die richtige Stadt an, bevor es mit der nächsten Frage weitergeht.

Da die Länder und Städte in Form von DATA-Zeilen vorliegen, kann dieses Programm jederzeit erweitert werden. Wenn Sie einen Sport-Quiz bevorzugen: Ändern Sie einfach die entsprechenden DATA-Zeilen. Wenn Sie die Zahl der Fragen erhöhen, vergessen Sie nicht, in Zeile 10 den Ausgangswert für N anzupassen. (F.Fenske/ef)

```
10 LET N=28:LET W=0:LET R=0:FOR I=1 TO N:R
   EAD X$,Y$                                 <178>
20 PRINT"{CLR,RVSON}HAUPTSTADT QUIZ 1988 B
   Y FRANK FENSKE":PRINT:PRINT               <238>
30 PRINT"WIE HEISST DIE HAUPTSTADT VON/DER
   ":PRINT:PRINT" ";X$;"?":PRINT             <187>
40 INPUT Z$ :PRINT                           <255>
50 IF Y$=Z$THEN LET R=R+1:PRINT"{RVSON}RIC
   HTIG ";Y$;"":FOR F=0 TO 500:NEXT F:PRIN
   T:NEXT I                                  <146>
60 IF Y$<>Z$THEN LET W=W+1:PRINT"{RVSON}FA
   LSCH ";Y$;"":FOR F=0 TO 500:NEXT F:PRIN
```

Listing 6. Prüfen Sie Ihre Erdkunde-Kenntnisse

```
   T:NEXT I                                <180>
70 PRINT"SIE WURDEN ";N;"MAL GEFRAGT":PRIN
   T"SIE GABEN ";R;" RICHTIGE ANTWORTEN"   <231>
80 PRINT"SIE GABEN ";W;" FALSCHE ANTWORTEN
   "                                       <053>
90 PRINT"{DOWN,RVSON}DAS WARS LEUTE{3SPACE
   }RUN ODER RESET"                        <130>
100 DATA"AUSTRALIEN", "CANBERRA":DATA"U.S.
    A.","WASHINGTON"                       <002>
110 DATA"JAPAN","TOKYO":DATA"TUERKEI","ANK
    ARA":DATA"SPANIEN","MADRID"            <222>
120 DATA"DEUTSCHLAND","BONN":DATA"CHINA","
    PEKING":DATA"AEGYPTEN","KAIRO"         <093>
130 DATA"U.D.S.S.R","MOSKAU":DATA"OESTERRE
    ICH","WIEN":DATA"ENGLAND","LONDON"     <252>
140 DATA"IRAN","TEHERAN":DATA"SCHWEDEN","S
    TOCKHOLM":DATA"DAENEMARK","KOPENHAGEN" <115>
150 DATA"ISRAEL","JERUSALEM":DATA"KANADA",
    "OTTAWA":DATA"LICHTENSTEIN","VADUZ"    <034>
160 DATA"THAILAND","BANGKOK":DATA"UNGARN",
    "BUDAPEST":DATA"LIBANON","BEIRUT"      <197>
170 DATA"ALGERIEN","ALGIER":DATA"CHILE","S
    ANTIAGO":DATA"CUBA","HAVANNA"          <111>
180 DATA"GRIECHENLAND","ATHEN":DATA"IRLAND
    ","DUBLIN":DATA"ITALIEN","ROM"         <121>
190 DATA"WER SCHRIEB DIESES PROGRAMM","FRA
    NK FENSKE"                             <030>
200 DATA"WANN WURDE DIESES PROGRAMM GESCHR
    IEBEN","1988"                          <224>
```

Listing 6. (Schluß)

22. Datasette richtig justiert

Jetzt ist Schluß mit »LOAD ERROR« beim Abspielen von fremden Kassetten.

Jeder, der eine Datasette besitzt, weiß, daß diese bei einem anderen C 64-Besitzer meist nicht mit derselben Tonkopfeinstellung justiert ist. Bekommt man eine heißerwartete Kassette mit Spielen oder anderen Programmen, womöglich noch im Turbo-Tape-Format, ist die Enttäuschung groß, wenn man feststellt, daß die Tonkopfjustierung von Aufnahme- und Wiedergabegerät abweicht.

»SYNCHRO JUSTAGE« (Listing 7) brennt einer Kassette ein 50-Herz-Synchronsignal auf, das nach der Übergabe der Kassette mit »SYNCHRO JUSTAGE« auf dem Bildschirm ausgewertet wird. Der Empfänger stellt nun mit der Einstellschraube seinen Tonkopf so ein, daß das Flackern des Bildschirmes minimal wird. Je weniger er flackert, desto besser hat man justiert. Wandert der Synchronbalken nach oben, läuft die Datasette bei der Wiedergabe schneller als bei der Aufnahme und umgekehrt. Dann gilt es, zusätzlich die Motorgeschwindigkeit zu überprüfen und optimal einzustellen, bis der schwarze Balken stillsteht.

Ich tausche nur noch Kassetten mit Synchron-Signal aus und kann keine Probleme mit meiner Datasette beklagen. Man muß zwar von Zeit zu Zeit seine Tonkopfeinstellung einer anderen anpassen, kann aber seinem Bekannten sagen, wie viele Umdrehungen seine Einstellschraube von der eigenen abweicht. (Reinhard Abdel-Hamid/ef)

```
10 FOR I=36864 TO 36935:READ A:S=S+A:POKE
   I,A:NEXT                                <054>
20 IF S<>6828 THEN PRINT"FEHLER IN DATA'S!
   ":STOP                                  <034>
30 PRINT"{CLR,DOWN,5RIGHT}SYNCHRO-TEST FUE
   R DATASETTEN"                           <041>
40 PRINT"{3DOWN,5RIGHT}RUECKSPULEN UND 'PL
   AY' ODER"                               <006>
```

Listing 7. »Synchro Justage«

```
50 PRINT"{6RIGHT}'PLAY'+'RECORD' DRUECKEN. <119>
60 PRINT"{2DOWN}SIGNAL{SPACE,RVSON}A{RVOFF
   }USWERTEN ODER{SPACE,RVSON}S{RVOFF}CHRE
   IBEN?' ";                               <123>
70 POKE 204,0:GET A$:IF A$="A"THEN SYS 368
   96                                      <188>
80 IF A$<>"S"THEN 70                       <174>
90 SYS 36864                               <126>
100 DATA 169,0,141,17,208,120,173,18,208,7
    4,74,74,74,41,8,9,7,133,1,141,24       <204>
110 DATA 212,74,74,74,141,32,208,76,6,144,
    234,120,169,7,133,1,169,0,141,17       <020>
120 DATA 208,141,32,208,173,13,220,41,16,2
    40,249,173,32,208,73,1,141,32,208      <046>
130 DATA 10,10,10,141,24,212,76,45,144,0,0
    ,1                                     <068>
```


Listing 7. (Schluß)

23. Betrifft: Joystick

Jeder Neuling unter den C 64-Fans wird sich früher oder später fragen, warum Commodore gleich zwei Joystick-Ports einbaute, es aber versäumte, das Basic des C 64 um eine Funktion zu bereichern, diese Joysticks auch abzufragen. Viele andere Computer haben einen »JOY(X)«-Befehl, der die Richtung angibt, in die der Joystick gerade gedrückt wird.

Auf dem C 64 wurden solche Abfragen bisher mit langwierigen IF-THEN-Sequenzen über die PEEK-Funktion realisiert. Es gibt aber einen sehr viel eleganteren Weg: die DEF FN-Funktion.

Wenn man nämlich am Anfang eines Basic-Programms definiert:

```
DEFFNJOY(X)=INT((LOG(255.5-(PEEK(56322-X)OR224)))/
LOG(2)+2)
```

so läßt sich über »PRINT FN JOY(x)« der Joystick abfragen. »x« gibt dabei an, ob man die Position von Joystick-Port 1 oder 2 wissen möchte. Es entsprechen: 1: Nullstellung, 2: oben, 3: unten, 4: links, 5: rechts und 6: Feuerknopf

Über eine »ON FN JOY(x) GOTO ...«-Anweisung ließe sich dann äußerst schnell in die entsprechenden Unterprogramme verzweigen. (Henning Zipf/ef)

24. Autostart mit Trick

Wenn ein Basic-Programm ein zweites Basic-Programm nachlädt, wird das zweite Programm automatisch gestartet. Das läßt sich sehr leicht zu einem Autostart für ein beliebiges Basic-Programm ausnutzen.

Sie benötigen dafür ein Vorprogramm, das nur aus zwei Zeilen besteht:

```
10 INPUT "WELCHES PROGRAMM ";D$
20 LOAD D$,8
```

Kennen Sie den Namen des Programms, das Sie laden und starten wollen, geben Sie ihn nach der Aufforderung ein. Es wird sofort geladen und automatisch gestartet. Sehr einfach wird es, wenn dieses Vorprogramm das erste auf der Diskette ist: Es kann mit »LOAD "*",8 geladen werden.

Dieser einfache Trick kann komfortabel ausgearbeitet werden, wenn beispielsweise das Directory zur besseren Übersicht zusätzlich eingelesen wird. Bei der Eingabe des Programmnamens dürfen Sie wie gewohnt den <*> als Platzhalter verwenden. (C.Burfeind/ef)

25. Geräusche von A bis Z

Durch Veränderung der Filterfrequenz und durch verschiedene Filter lassen sich einfache Geräusche erzeugen.
Zeile 10 S=54272:Rem Basisregister
Zeile 20 FORL=0TO24:POKES+L,0:NEXT

	Zeile 30	Zeile 40	Zeile 50	Zeile 60	Zeile 70	Zeile 80	Zeile 90	Zeile 100
Geräusch:	Frequenz	Hall	Grenz-frequenz	Resonanz	Pass	Wellenform	Schleife und POKE	Warteschleife und GOTO
Schuß	POKE S+0,0: POKE S+1,18	POKE S+5,1*16+11	POKE S+22,110	POKE S+23,15*16+3	POKE S+24,5*16+15	POKE S+4,0: POKE S+4,129	FORJ=1TO 255: POKE S+0,J: NEXT	FORA=1TO 1000: NEXT:GOTO80
Explosion	POKE S+0,0: POKE S+1,6	POKE S+5,2*16+13	POKE S+22,100	POKE S+23,15*16+3	POKE S+24,3*16+15	POKE S+4,0: POKE S+4,129	FORJ=1TO 100: POKE S+0,J: NEXT	FORA=1TO 4000: NEXT:GOTO80
Uhrenschlag	POKE S+0,0: POKE S+1,6	POKE S+5,1*16+10	POKE S+22,110	POKE S+23,15*16+3	POKE S+24,1*16+15	POKE S+4,0: POKE S+4,17	FORJ=1TO 255: POKE S+0,7: NEXT	FORA=1TO 500: NEXT:GOTO80
Brandung	POKE S+0,0: POKE S+1,40	POKE S+5,10*16+12	POKE S+22,0	POKE S+23,0	POKE S+24,0*16+15	POKE S+4,0: POKE S+4,129	FORJ=1TO 255: POKE S+0,J: NEXT	FORA=1TO 3500: NEXT:GOTO80

Um einen Schuß zu erzeugen, muß das Programm zum Beispiel so aussehen:

```
 10 S=54272
 20 FORL=0TO24:POKE S + L,0: NEXT
 30 POKE S + 0,0 : POKE S + 1,18
 40 POKE S + 5,1 * 16 + 11
 50 POKE S + 22, 110
 60 POKE S + 23, 15 * 16 +3
 70 POKE S + 24, 5 * 16 + 15
 80 POKE S + 4, 0 : POKE S + 4, 129
 90 FORJ = 1 TO 255 : POKE S + 0,J : NEXT
100 FORA = 1 T 1000 : NEXT : GOTO 80
```

Experimentieren Sie ein wenig mit den verschiedenen Werten, um neue und interessante Soundeffekte zu erzeugen. Als Unterprogramm lassen Sie sich leicht in eigene Programme einbinden.

(Jürgen Hüsgen/ef)

26. Professionelles Auswahlmenü

Wenn man ein Programm schreibt, in dem der Benutzer aus mehreren Programmfunktionen auswählen kann, kommt man um die Erstellung eines Menüs nicht hinweg. Dieses Menü kann zum Beispiel so aussehen: Die verschiedenen Wahlmöglichkeiten bekommen eine Nummer, und der Benutzer gibt beim Programmablauf die gewünschte Nummer ein. Das Programm verzweigt dann entsprechend der Eingabe.

Eine optisch schon wesentlich ansprechendere Methode sehen Sie in Listing 8. Tippen Sie dieses Listing bitte ein und beachten dabei unsere Eingabehinweise auf Seite 159.

Das Programm ist nur ein Beispiel, um das Funktionsprinzip zu erklären. Es enthält sechs Menüpunkte, deren Texte in der Zeile 90 abgelegt sind. Mittels der Cursor-Tasten kann man sich einen Menüpunkt aussuchen und durch Druck auf die <RETURN>-Taste aktivieren. In Zeile 80 könnte zum Beispiel eine »ON A GOTO 200,300,...«-Verzweigung erfolgen. (Ulrich Wichelhaus/ef)

```
10 A=1                                          <183>
15 PRINT"{CLR,4DOWN}"                           <002>
20 FOR I=1 TO 6                                 <249>
30 READ M$(I)                                   <211>
40 PRINT TAB(10)M$(I)                           <150>
50 NEXT                                         <060>
60 GOSUB 100                                    <004>
70 PRINT"{CLR}SIE WAEHLTEN PUNKT"A              <176>
80 END:REM HIER KOENNTE DANN VERZWEIGUNG E
   RFOLGEN                                      <007>
90 DATA PUNKT 1,PUNKT 2,PUNKT 3,PUNKT 4,PU
   NKT 5,PUNKT 6                                <109>
100 REM ******************************          <238>
105 REM HIER BEGINNT DIE AUSWAHLROUTINE         <135>
110 POKE 214,A+4:POKE 211,10:SYS 58640:PRI
    NT"{RVSON}"M$(A)                            <163>
120 GET X$:IF X$=""THEN 120                     <189>
130 IF X$=CHR$(13)THEN RETURN                   <111>
140 IF X$="{DOWN}"THEN 180                      <029>
150 POKE 214,A+4:POKE 211,10:SYS 58640:PRI
    NT M$(A)                                    <055>
160 A=A-1:IF A<1 THEN A=1                       <118>
170 GOTO 110                                    <114>
180 POKE 214,A+4:POKE 211,10:SYS 58640:PRI
    NT M$(A)                                    <085>
190 A=A+1:IF A>6 THEN A=6                       <023>
200 GOTO 110                                    <144>
```

Listing 8. Ein professionelles Auswahlmenü

27. »Der Gänsefüßchenmodus«

Angenommen, Sie schreiben gerade an einem Basic-Programm. Mittels einer PRINT-Anweisung möchten Sie einen Text ausdrucken. Beispiel: `PRINT "DIS IST EIN TEXT...`

Hoppla, »dies« schreibt man natürlich mit »ie«. Also mit dem Cursor wieder nach links und den Fehler ausbessern. Aber halt! Da erscheinen plötzlich so komisch reverse Zeichen, wenn man eine der Cursor-Tasten drückt! Ist mein C 64 defekt? Keine Sorge, der C 64 ist völlig in Ordnung. Die Erklärung lautet »Gänsefüßchenmodus«. Angenommen, Sie möchten, daß der Computer an einer bestimmten Stelle

im Programm den Bildschirm löscht. Tippen Sie einmal folgendes ein:

```
10 PRINT"
```

und drücken dann die Taste für »Bildschirm löschen«. Was passiert? Der Bildschirm wird nicht sofort gelöscht, sondern nach dem Anführungszeichen erscheint ein »reverses Herz«. Dies ist der »Steuercode« für »Bildschirm löschen«. Wenn Sie jetzt »RUN« eintippen (vorher die RETURN-Taste drücken), wird der Bildschirm tatsächlich gelöscht. Wir halten fest: Sobald Sie in einer Zeile das erste Mal das Anführungszeichen eingeben, werden alle darauffolgenden Cursorbewegungen und Direktbefehle nicht mehr ausgeführt, sondern in einen »computerlesbaren« Code übersetzt. So lassen sich gezielt bestimmte Cursorpositionen ansteuern.

Übrigens: Alle unsere Basic-Listings sind von diesen, auf Papier nur schwer zu unterscheidenden Steuerzeichen befreit. Wenn Sie in der 64'er zum Beispiel folgendes sehen, dann dürfen Sie die Anweisungen in den geschweiften Klammern nicht ausführen, sondern müssen die entsprechende Taste drücken.

Beispiel: `10 PRINT "{CLR,3DOWN}TEST"`

Also zuerst die »Bildschirm löschen«-Taste (CLR) und dann dreimal die »Cursor nach unten«-Taste (DOWN) drücken. Möchten Sie den Gänsefüßchen-Modus ausschalten, kann dies auf zwei Arten geschehen. Erstens: Wir drücken die SHIFT und die RETURN-Taste.

Oder zweitens: Wir drücken erneut das Anführungszeichen. Beide Methoden schalten den Gänsefüßchenmodus aus und erlauben wieder normale Cursorbewegungen. Experimentieren Sie ruhig einmal mit diesem Modus. Sie wissen ja: Übung macht den Meister. (Thomas Röder/ef)

28. C64 wird schneller

Es ist allgemein bekannt, daß das C 64-Basic nicht gerade das schnellste ist. Durch einen kleinen Trick läßt es sich aber um zirka 6 Prozent beschleunigen. Durch »POKE 53265, PEEK(53265) AND 239« wird der Bildschirm und damit auch der Video-Prozessor abgeschaltet. Dieser hat nämlich die unangenehme Eigenschaft, bei jedem Speicherzugriff (»welche Zeichen müssen nun auf dem Bildschirm dargestellt werden?«) den Prozessor kurzerhand anzuhalten. Mit dem oben genannten POKE wird nun der Video-Prozessor an dieser unliebsamen Angewohnheit gehindert. Ab jetzt werden PRINT-Anweisungen zwar ausgeführt, sind aber auf dem Monitor oder Fernseher nicht mehr sichtbar. Erst durch »POKE 53265, PEEK(53265) OR 16« ist der Bildschirminhalt wieder sichtbar. Sinnvoll ist dieses Verfahren zum Beispiel bei komplizierten mathematischen Berechnungen. Achtung: Wenn das Programm mit einer Fehlermeldung aussteigt, ist diese natürlich auch nicht mehr sichtbar. Man sollte also während der Aus-Phase zum Beispiel alle fünf Sekunden einen Signalton geben oder die Bildschirmfarbe (»POKE 53280,Farbe«) ändern. So hat man immer die Gewähr, daß das Programm noch läuft. (Michael Rauh/ef)

29. Der Mini-Autostart

Wer zu faul ist, jedesmal »RUN« einzutippen, wenn ein Programm von der Floppy geladen wurde, kann jetzt aufatmen: Ein kleiner Trick macht dies automatisch.

Wie Sie vielleicht wissen, bewirkt ein SHIFT-RUN/STOP-Tastendruck ein Laden des nächsten Programms von Datasette mit automatischem RUN danach. Diesen Umstand können sich die Floppy-Besitzer zunutze machen.

Wenn Sie ein Programm laden möchten, tippen Sie ganz normal »LOAD "Programmname",8« ein und setzen dahinter noch einen Doppelpunkt. Drücken Sie nun <SHIFT-RUN/STOP>. Auf dem Bildschirm erscheint hinter dem Doppelpunkt noch ein LOAD-Befehl (dieser wird vom C 64 jedoch ignoriert). Wenn das Programm fertig geladen ist, führt der Computer ein RUN aus. (Bernd Roggendorf/ef)

30. Was tun bei »out of memory«?

Hatten Sie auch schon mal während der Arbeit mit dem C 64 plötzlich aus heiterem Himmel einen »out of memory error«? Abgesehen davon, daß Sie wirklich ein äußerst langes Programm im Speicher haben könnten, taucht diese Meldung auch nach dem Laden von vielen Maschinenprogrammen auf (diese erkennen Sie daran, daß sie nicht mit RUN, sondern mit einem SYS-Befehl gestartet werden). Die Fehlermeldung kommt meistens dann, wenn Sie nach dem Laden entweder ein Basic-Programm mit RUN starten, oder aber ein weiteres Programm laden möchten.

Warum taucht dieser Fehler auf? Wenn Sie zum Beispiel eine Variable mit einem Wert belegen (A=10), dann legt der C 64 diese Zahl 10 direkt hinter dem aktuellen (Basic-)Programm im Speicher ab. Dazu muß er aber wissen, wo das Programm aufhört. Zu diesem Zweck gibt es einen sogenannten »Zeiger«, der die letzte vom Programm belegte Adresse angibt. Der Zeiger wird aktualisiert, wenn Sie das Programm in seiner Länge verändern, also zum Beispiel eine Zeile hinzuschreiben oder löschen. Der Zeiger wird aber auch gesetzt, wenn Sie ein Programm von Diskette oder Kassette laden.

Maschinenprogramme werden vom Programmierer meistens in Speicherbereiche gelegt, die von Basic-Programmen aus normalerweise nicht überschrieben werden. Diese Bereiche haben sehr hohe Adressen, auf die der eben beschriebene Programmende-Zeiger nach dem Laden gesetzt wird. Wenn Sie jetzt einen Befehl eingeben, der eine Variable definiert, erkennt der C64, daß hinter dem Programmende-Zeiger bereits ein Speicherbereich liegt, der von einem Basic-Programm nicht benutzt werden darf.

Ein Beispiel: Basic-Programme haben im allgemeinen den Speicherbereich von Adresse 2049 bis Adresse 40959 zur Verfügung. Maschinenprogramme verwenden jedoch häufig den etwas höher liegenden Bereich zwischen Adresse 49152 und Adresse 53247. Wenn ein Programm in diesen hohen Bereich geladen, also der Zeiger zum Beispiel auf Adresse 51234 gesetzt wird, kommt es beim späteren Belegen einer Variablen zum »out of memory error«.

Auch beim Laden von Programmen wird übrigens eine Variable belegt: Der C64 merkt sich den Namen des zu ladenden Programms am Ende des Basic-Speichers. Vorher wird jedoch anhand des Programmende-Zeigers überprüft, ob dafür überhaupt noch Platz ist. In unserem Beispiel mit der Adresse 51234 ist dies nicht der Fall. Also kommt es auch beim Nachladen von Programmen zum »out of memory error«.

Doch was tun? Geben Sie einfach NEW ein. Keine Angst: Dadurch wird das Maschinenprogramm nicht gelöscht. Lediglich der Programmende-Zeiger wird auf Adresse 2051 gesetzt, also festgelegt, daß sich kein Basic-Programm mehr im Speicher befindet. (Thomas Röder/ef)

Checksummer V3 und MSE

Diese beiden Programme sind unentbehrlich beim Abtippen unserer Listings. Sie helfen, Tippfehler vor allem bei Maschinenprogrammen zu vermeiden und sparen eine Menge Zeit.

Nobody is perfect. Jeder Computer-Fan, egal ob blutiger Anfänger oder ausgefuchster Profi, macht beim Abtippen von Programmen Tippfehler. Diese Fehler später zu finden, kann ein langwieriges Unterfangen sein.

Deshalb haben wir für Sie die Programme »Checksummer V3« und »MSE« (MaschinenSpracheEditor) entwickelt. Der Checksummer ist für Basic-Programme und der MSE für Maschinensprache-Listings zuständig.

Der Checksummer

Zuerst einmal müssen Sie das Checksummer-Programm (siehe Listing 1) abtippen. Dabei sollten Sie äußerst sorgfältig vorgehen, vor allem bei den Zahlen in den DATA-Zeilen 20 bis 30. Wenn Sie trotzdem noch einen Tippfehler gemacht haben, meldet sich das Programm später mit einem entsprechenden Hinweis. Wenn Sie fertig sind, speichern Sie das Programm auf Diskette oder Kassette.

Jetzt geht es los:

1. Starten Sie den Checksummer durch die Eingabe von »RUN« und das Drücken der RETURN-Taste.
2. Wenn die Meldung »Checksummer aktiviert...« auf dem Bildschirm erscheint, haben Sie keinen Tippfehler gemacht und der Checksummer ist nun eingeschaltet.
3. Zum Löschen des Basic-Programms geben Sie bitte »NEW« ein. Keine Angst, der Checksummer selbst wird dadurch nicht gelöscht.
4. Nun können wir den Checksummer testen. Geben Sie bitte folgende Zeile ein und drücken Sie die RETURN-Taste:

1 REM

In der linken oberen Bildschirmecke sehen Sie nun die Prüfsumme über die eben eingegebene Basic-Zeile. Sie muß <63> lauten. Dem Checksummer ist es übrigens egal, ob Sie »1 REM« oder »1REM« eintippen. Nur innerhalb von Anführungszeichen ist die richtige Anzahl an Leerzeichen wichtig. Diese Prüfsummen erscheinen (sofern Sie den Checksummer eingeschaltet haben) immer dann, wenn Sie eine Basic-Zeile eintippen und dann die RETURN-Taste drücken. In der 64'er finden Sie die Prüfsumme immer am Ende jeder Programmzeile.

```
10 PRINT"CHECKSUMMER FUER C 64"
11 PRINT:PRINT"EINEN MOMENT, BITTE ..."
12 FOR I=828 TO 864:READ A:POKE I,A:PS=PS+
   A:NEXT I
13 IF PS<>5765 THEN PRINT"TIPPFEHLER IN DE
   N ZEILEN 20 BIS 22":END
14 SYS 828:PS=0:FOR I=58464 TO 58583:READ
   A:POKE I,A:PS=PS+A:NEXT I
15 IF PS<>16147 THEN PRINT"TIPPFEHLER IN D
   EN ZEILEN 22 BIS 30":END
16 POKE 1,53:POKE 42289,96:POKE 42290,228
17 PRINT"CHECKSUMMER AKTIVIERT."
18 PRINT:PRINT" AUSSCHALTEN : POKE1,55 ODE
   R"SPC(27)"<RUN/STOP+RESTORE>"
19 PRINT:PRINT" ANSCHALTEN : POKE1,53"
20 DATA 169,0,133,254,162,1,189,93,3,133,2
   55,160,0,177,254
21 DATA 145,254,136,208,249,230,255,165,25
   5,221,95,3,208,238,202
22 DATA 16,230,96,160,224,192,0,160,2,169,
   0,170,133,254,177
23 DATA 95,240,40,201,32,208,3,200,208,245
   ,133,255,138,41,7
24 DATA 170,240,14,72,165,255,24,42,105,0,
   202,208,249,133,255
25 DATA 104,170,232,165,255,24,101,254,133
   ,254,76,111,228,192,4
26 DATA 48,219,198,214,165,214,72,162,3,16
   9,32,157,1,4,189
27 DATA 212,228,32,210,255,208,12,0,92,72,
   32,201,255,170,104
28 DATA 144,1,138,96,202,16,228,166,254,16
   9,0,32,205,189,169
29 DATA 62,32,210,255,104,133,214,32,108,2
   29,169,141,32,210,255
30 DATA 76,128,164,9,60,18,19
```

© 64'er

Listing 1. Der »Checksummer 64 V3« für Basic-Listings

```
5 PRINT CHR$(14)                                  <242>
10 PRINT"{CLR}"                                   <254>
20 PRINT"Q*****************************W"         <130>
30 PRINT"{4DOWN,2SPACE}TEST{SPACE,BLUE,6SP
   ACE}"                                          <022>
40 PRINT"BBBBBBBBBBBBBBBBBBBBBBBBBBBBB"           <108>
```

© 64'er

Bild 1. Die Bedeutung der Steuerzeichen wird im nachfolgenden Text erklärt

In Zeile 10 müssen Sie nach den Anführungszeichen die Tasten <SHIFT CLR/HOME> drücken und nicht die Klammern mit dem Wort CLR eingeben. In Zeile 20 drücken Sie nach den Anführungszeichen die CBM-Taste und den Buchstaben <Q>, gefolgt von mehreren SHIFT- und Stern-Tasten und zum Schluß die CBM-Taste und den Buchstaben <W>. In Zeile 30 ist es viermal die CURSOR-abwärts-Taste, gefolgt von zweimaliger Leertaste, dann <SHIFT T> und normal EST, zum Schluß noch einmal die Leertaste, die Farbtaste Blau <CTRL 7> und sechsmal die Leertaste. Zeile 40 besteht lediglich aus mehreren Grafikzeichen, die mit der CBM-Taste und <B> erzeugt werden.

CTRL steht für Control-Taste, so bedeutet {CTRL+A}, daß Sie die Control-Taste und die Taste »A« drücken müssen. Im folgenden steht:

Befehl	Bedeutung	Befehl	Bedeutung	Befehl	Bedeutung
{DOWN}	Taste neben rechtem Shift, Cursor unten	{SPACE}	Leertaste	{RVSON}	Control-Taste & 9
{UP}	Shift-Taste & Taste neben rechtem Shift; Cursor hoch	{SHIFT-Space}	Shift-Taste & Leertaste	{RVOFF}	Control-Taste & 0
{CLR}	Shift-Taste & 2. Taste ganz rechts oben	{F1} bis {F8}	Funktionstasten	{ORANGE}	Commodore-Taste & 1
{INST}	Shift-Taste & Taste ganz rechts oben	{RETURN}	Return-Taste	{BROWN}	Commodore-Taste & 2
{HOME}	2. Taste von ganz rechts oben	{BLACK}	Control-Taste & 1	{LIG.RED}	Commodore-Taste & 3
{DEL}	Taste ganz rechts oben	{WHITE}	Control-Taste & 2	{GREY 1}	Commodore-Taste & 4
{RIGHT}	Taste ganz rechts unten	{RED}	Control-Taste & 3	{GREY 2}	Commodore-Taste & 5
{LEFT}	Shift-Taste & Taste unten rechts	{CYAN}	Control-Taste & 4	{LIG.GREEN}	Commodore-Taste & 6
		{PURPLE}	Control-Taste & 5	{LIG.BLUE}	Commodore-Taste & 7
		{GREEN}	Control-Taste & 6	{GREY 3}	Commodore-Taste & 8
		{BLUE}	Control-Taste & 7		
		{YELLOW}	Control-Taste & 8		

Tabelle 1. Die Steuerbefehle in den Listings

Diese Zahlen dürfen Sie NICHT mit abtippen.

Als Beispiel sehen Sie Bild 1. Am rechten Rand jeder Spalte sehen Sie die Prüfsummen in eckigen Klammern.

Damit sind wir beim zweiten wichtigen Punkt: Sehen Sie sich die Zeile 240 von Listing 2 genauer an. Nach dem ersten Anführungszeichen nach dem PRINT-Befehl sehen Sie eine geschweifte Klammer { }. Immer, wenn Sie in einem unserer Listings diese Klammern sehen, dürfen Sie das, was innerhalb der Klammern steht, nicht eintippen. Sie müssen die entsprechende Taste drücken. Beispiel:
10 PRINT "{CLR}"
bedeutet: Nach dem Anführungszeichen die »Bildschirmlöschen«-Taste drücken (<SHIFT CLR/HOME>). In Tabelle 1 sehen Sie eine Zusammenfassung aller möglichen Steuertasten mit dem entsprechenden Klartext.

Weiterhin sehen Sie in Bild 1 (Bedeutung der Steuerzeichen) in Zeile 30 ein unterstrichenes »T« nach der Klammer. Das bedeutet, daß Sie ein »T« zusammen mit der SHIFT-Taste drücken müssen, also <SHIFT T>. Wenn ein Zeichen »überstrichen« ist, müssen Sie dieses zusammen mit der CBM-Taste eingeben. Die CBM-Taste befindet sich ganz links unten auf der Tastatur und hat die Aufschrift »C=«.

```
100 REM DIESES PROGRAMM ERZEUGT DEN              <210>
110 REM MSE V1.1 AUF DISKETTE.                   <039>
120 REM BESITZER EINER DATASETTE                 <178>
130 REM MUESSEN DIE '8' AM ENDE VON              <145>
140 REM ZEILE 343 IN EINE '1' AENDERN!           <176>
150 REM                                          <212>
230 IF PEEK(44)<>32 THEN PRINT"{CLR}SIE HA
    BEN VERGESSEN, DIE POKES EINZUGE- BEN!
    ":END                                        <050>
240 PRINT"{CLR}";:DIM H(75):FOR I=0 TO 9         <042>
250 H(48+I)=I:H(65+I)=I+10:NEXT:Z=1000           <136>
260 FOR I=2048 TO 3755 STEP 20:PRINT"{HOME
    }ICH LESE ZEILE:"Z                           <253>
261 FOR N=0 TO 19:READ A$:IF LEN(A$)<>2 TH
    EN 900                                       <062>
262 IF PEEK(63)+PEEK(64)*256<>Z THEN 800         <011>
270 H=ASC(LEFT$(A$,1)):L=ASC(RIGHT$(A$,1))       <199>
280 D=H(H)*16+H(L):S=S+D:POKE I+N,D              <165>
290 NEXT:READ V:IF S<>V THEN 900                 <139>
300 S=0:Z=Z+1:NEXT:R=PEEK(2111):H=PEEK(210
    6)                                           <126>
301 POKE 53280,R:POKE 53281,H:POKE 646,R:P
    RINT"{CLR}DIE DATA-ZEILEN SIND FEHLERF
    REI!"                                        <080>
302 PRINT"SIE KOENNEN NUN DIE FARBEN DES M
    SE"                                          <209>
303 PRINT"EINSTELLEN.":PRINT"{2DOWN,SPACE,
    RVSON}DRUECKEN SIE <1>, <2> ODER <9>         <205>
304 PRINT"{DOWN,2SPACE}<1> - RAHMEN-/SCHRI
    FTFARBE                                      <013>
305 PRINT"{2SPACE}<2> - HINTERGRUNDFARBE         <233>
306 PRINT"{DOWN,2SPACE}<9> - FARBEN UEBERN
    EHMEN                                        <158>
307 PRINT"{2DOWN}FARBE <1>:"R:PRINT"FARBE
    <2>:"H                                       <066>
308 GET A:IF A=0 THEN 308                        <210>
309 IF A=1 THEN R=(R+1)AND 15                    <098>
310 IF A=2 THEN H=(H+1)AND 15                    <086>
311 IF A=9 THEN 340                              <217>
312 GOTO 301                                     <034>
340 POKE 2106,H:POKE 2111,R                      <153>
342 POKE 631,19:POKE 632,13:POKE 198,2           <135>
343 PRINT"{CLR}SAVE"CHR$(34)"MSE V1.1"CHR$
    (34)",8                                      <091>
344 POKE 43,1:POKE 44,8:POKE 45,172:POKE 4
    6,14:END                                     <140>
800 PRINT"{CLR,RVSON}SIE HABEN ZEILE"Z"{LE
    FT,SPACE}VERGESSEN:":A=PEEK(646)AND 15       <124>
810 POKE 646,PEEK(53281)AND 15:PRINT"LIST"
    Z-2"-"Z+2:POKE 646,A                         <224>
820 GOTO 920                                     <082>
900 PRINT"{CLR,RVSON}SIE HABEN EINEN TIPPF
    EHLER GEMACHT:":A=PEEK(646)AND 15            <154>
910 POKE 646,PEEK(53281)AND 15:PRINT"LIST"
    Z:POKE 646,A                                 <173>
920 POKE 631,19:POKE 632,17:POKE 633,13:PO
    KE 198,3:END                                 <126>
1000 DATA 00,0B,08,0A,00,9E,32,30,36,31,00
     ,00,00,A2,08,A9,36,85,A4,A9, 1247           <119>
1001 DATA 08,85,A5,A9,00,85,A6,A9,B0,85,A7
     ,A0,00,B1,A4,91,A6,C8,D0,F9, 2888           <054>
1002 DATA E6,A5,E6,A7,CA,D0,F2,A9,36,85,01
     ,4C,00,B0,20,D1,B1,A9,00,8D, 2781           <096>
1003 DATA 21,D0,A9,0F,8D,20,D0,8D,86,02,A0
     ,B3,A9,74,20,FF,B1,A0,B3,A9, 2679           <089>
1004 DATA B9,20,FF,B1,A0,00,20,CF,FF,99,01
     ,02,C8,C9,0D,D0,F5,88,F0,D2, 2912           <217>
1005 DATA C0,11,90,02,A0,10,8C,00,02,20,EA
     ,B1,A0,B3,A9,CF,20,FF,B1,20, 2327           <045>
1006 DATA 8E,B4,85,FC,85,62,20,8E,B4,85,FB
     ,85,61,20,A7,B4,D0,20,A0,B3, 2864           <199>
1007 DATA A9,E5,20,FF,B1,20,8E,B4,85,60,20
     ,8E,B4,85,5F,20,A7,B4,D0,0A, 2624           <091>
1008 DATA A5,61,C5,5F,A5,62,E5,60,90,06,20
     ,43,B3,4C,3A,B0,A9,AA,A0,00, 2379           <167>
1009 DATA EA,EA,E6,FB,D0,02,E6,FC,20,3F,B2
     ,90,EF,4C,FB,B4,A2,02,86,58, 3190           <041>
1010 DATA A9,A6,A0,9D,20,F2,B1,20,E4,FF,F0
     ,FB,C9,30,90,0C,C9,47,B0,08, 2970           <231>
1011 DATA C9,3A,90,0B,C9,41,B0,07,C9,14,D0
     ,0F,4C,0B,B1,20,D2,FF,A6,58, 2322           <121>
1012 DATA 95,F7,C6,58,D0,D2,60,AE,8D,02,F0
     ,26,C9,0C,D0,03,4C,0B,B6,C9, 2685           <057>
1013 DATA 13,D0,03,4C,8B,B5,C9,0D,D0,03,4C
     ,BA,B4,C9,10,D0,03,4C,68,B5, 2282           <225>
1014 DATA C9,0E,D0,06,20,5F,B4,4C,64,B1,4C
     ,92,B0,A5,F9,20,02,B1,0A,0A, 2132           <208>
1015 DATA 0A,0A,85,F9,A5,F8,20,02,B1,05,F9
     ,60,C9,3A,90,02,69,08,29,0F, 1950           <092>
1016 DATA 60,A6,59,E0,08,90,1F,A6,58,E0,02
     ,B0,06,20,D2,FF,4C,8E,B0,C6, 2509           <188>
1017 DATA 59,A0,14,A9,92,20,F2,B1,CA,D0,FA
     ,84,57,68,68,4C,8B,B1,A6,D3, 2891           <197>
1018 DATA E0,08,B0,03,4C,92,B0,20,D2,FF,A6
     ,58,E0,02,90,09,C6,59,20,D2, 2468           <049>
1019 DATA FF,C6,58,D0,F9,4C,8E,B0,48,4A,4A
     ,4A,4A,20,59,B1,68,29,0F,C9, 2419           <035>
1020 DATA 0A,90,02,69,06,69,30,4C,D2,FF,A2
     ,FC,9A,20,D1,B1,20,48,B2,20, 2261           <073>
1021 DATA EA,B1,20,9F,B2,A5,FC,20,4E,B1,A5
     ,FB,20,4E,B1,20,ED,B1,A9,3A, 2860           <148>
1022 DATA A0,20,20,F2,B1,A9,00,85,59,20,8E
     ,B0,20,ED,B1,A4,59,20,EF,B0, 2530           <233>
1023 DATA 91,FB,C8,84,59,C0,08,90,EC,20,10
     ,B2,A9,12,20,D2,FF,20,8E,B0, 2657           <105>
1024 DATA 20,EF,B0,C5,FF,F0,0D,20,43,B3,A9
     ,14,A0,14,20,F2,B1,4C,A2,B1, 2665           <034>
1025 DATA A9,92,20,D2,FF,20,33,B2,20,E0,B2
     ,20,3F,B2,90,9F,4C,8B,B5,A9, 2648           <123>
1026 DATA 93,20,D2,FF,A2,00,A9,03,9D,00,D8
     ,9D,00,D9,9D,00,DA,9D,00,DB, 2476           <237>
1027 DATA E8,D0,EF,60,A9,0D,2C,A9,20,4C,D2
     ,FF,20,D2,FF,98,4C,D2,FF,20, 2965           <160>
1028 DATA E4,FF,F0,FB,60,84,5D,85,5C,A0,00
     ,B1,5C,F0,06,20,D2,FF,C8,D0, 3100           <077>
1029 DATA F6,60,A5,FB,85,5A,A0,00,84,5B,B1
     ,FB,18,65,5A,85,5A,90,02,E6, 2606           <156>
1030 DATA 5B,06,5A,26,5B,C8,C0,08,90,EC,A5
     ,5A,65,5B,85,FF,60,18,A5,FB, 2467           <219>
1031 DATA 69,08,85,FB,90,02,E6,FC,60,A5,FB
     ,C5,5F,A5,FC,E5,60,60,A0,B3, 3106           <183>
1032 DATA A9,FB,20,FF,B1,A0,01,B9,00,02,20
     ,D2,FF,CC,00,02,C8,90,F4,A9, 2692           <098>
1033 DATA 14,ED,00,02,AA,20,ED,B1,CA,D0,FA
     ,A5,62,20,4E,B1,A5,61,20,4E, 2457           <060>
1034 DATA B1,20,ED,B1,A5,60,20,4E,B1,A5,5F
     ,20,4E,B1,EA,EA,EA,EA,EA,EA, 3122           <190>
1035 DATA EA,EA,24,5E,10,01,60,A9,12,20,D2
     ,FF,A2,28,20,ED,B1,CA,D0,FA, 2703           <087>
1036 DATA A9,92,4C,D2,FF,A5,D6,C9,16,B0,01
     ,60,A9,A0,85,A4,A9,78,85,A6, 2945           <204>
1037 DATA A9,04,85,A5,85,A7,A2,13,A0,27,B1
     ,A4,91,A6,88,10,F9,CA,F0,19, 2671           <208>
1038 DATA 18,A5,A4,69,28,85,A4,90,02,E6,A5
     ,18,A5,A6,69,28,85,A6,90,E0, 2503           <251>
1039 DATA E6,A7,4C,B6,B2,A9,91,4C,D2,FF,A9
     ,0F,8D,18,D4,A9,00,8D,05,D4, 2776           <000>
1040 DATA A9,F7,8D,06,D4,A9,11,8D,04,D4,A9
     ,32,8D,01,D4,A9,00,8D,00,D4, 2413           <126>
1041 DATA A0,80,20,09,B3,A9,10,8D,04,D4,60
     ,A2,FF,CA,D0,FD,88,D0,F8,60, 2914           <240>
1042 DATA A9,0F,8D,18,D4,A9,2D,8D,05,D4,A9
     ,A5,8D,06,D4,A9,21,8D,04,D4, 2385           <119>
1043 DATA A9,07,8D,01,D4,A9,05,8D,00,D4,A0
```

```
     ,FF,2Ø,Ø9,B3,A9,2Ø,8D,Ø4,D4, 225Ø    <Ø78>
1Ø44 DATA A9,ØØ,8D,Ø1,D4,8D,ØØ,D4,6Ø,38,2Ø
     ,FØ,FF,8A,48,98,48,18,AØ,Ø6, 2179    <175>
1Ø45 DATA A2,18,2Ø,FØ,FF,AØ,B4,A9,ØA,2Ø,FF
     ,B1,2Ø,12,B3,2Ø,E4,FF,FØ,FB, 2931    <Ø93>
1Ø46 DATA A2,1D,A9,14,2Ø,D2,FF,CA,DØ,FA,68
     ,A8,68,AA,18,4C,FØ,FF,ØD,ØD, 27Ø4    <Ø88>
1Ø47 DATA ØD,2Ø,2Ø,2Ø,2Ø,2Ø,2Ø,2Ø,4D,41,53
     ,43,48,49,4E,45,4E,53,5Ø,52, 1144    <216>
1Ø48 DATA 41,43,48,45,2Ø,2D,2Ø,45,44,49,54
     ,4F,52,2Ø,ØD,ØD,2Ø,2Ø,2Ø,2Ø, 1Ø23    <Ø38>
1Ø49 DATA 2Ø,2Ø,2Ø,2Ø,56,4F,4E,2Ø,4E,2E,4D
     ,41,4E,4E,2Ø,26,2Ø,44,2E,57, 1128    <2Ø6>
1Ø5Ø DATA 45,49,4E,45,43,4B,ØØ,ØD,ØD,ØD,2Ø
     ,2Ø,2Ø,5Ø,52,4F,47,52,41,4D, 11Ø2    <117>
1Ø51 DATA 4D,4E,41,4D,45,2Ø,3A,2Ø,ØØ,ØD,ØD
     ,2Ø,2Ø,2Ø,53,54,41,52,54,41, 1Ø73    <Ø95>
1Ø52 DATA 44,52,45,53,53,45,2Ø,3A,2Ø,24,ØØ
     ,ØD,ØD,2Ø,2Ø,2Ø,45,4E,44,41, 1Ø14    <129>
1Ø53 DATA 44,52,45,53,53,45,2Ø,2Ø,2Ø,3A,2Ø
     ,24,ØØ,92,Ø1,Ø1,5Ø,52,4F,47, 1138    <228>
1Ø54 DATA 52,41,4D,4D,2Ø,3A,2Ø,ØØ,12,2Ø,2Ø
     ,2A,2A,2A,2Ø,46,41,4C,53,43, 1Ø24    <Ø27>
1Ø55 DATA 48,45,2Ø,45,49,4E,47,41,42,45,2Ø
     ,2A,2A,2A,2Ø,2Ø,92,ØØ,ØD,ØD, 1Ø58    <Ø98>
1Ø56 DATA 2A,2A,2A,2Ø,45,4E,44,45,2Ø,2A,2A
     ,2A,ØØ,13,Ø1,2Ø,2Ø,12,44,92, 916     <153>
1Ø57 DATA 49,53,4B,2Ø,4F,44,45,52,2Ø,12,54
     ,92,41,5Ø,45,ØD,ØØ,13,2Ø,2Ø, 1151    <Ø35>
1Ø58 DATA 49,2F,4F,2Ø,2D,2Ø,46,45,48,4C,45
     ,52,ØØ,2Ø,D1,B1,2Ø,48,B2,AØ, 16Ø6    <Ø12>
1Ø59 DATA B3,A9,CF,2Ø,FF,B1,2Ø,8E,B4,85,FC
     ,2Ø,8E,B4,85,FB,C5,61,A5,FC, 32Ø7    <251>
1Ø6Ø DATA E5,62,9Ø,23,A5,FB,C5,5F,A5,FC,E5
     ,6Ø,BØ,19,2Ø,A7,B4,DØ,14,6Ø, 286Ø    <112>
1Ø61 DATA 2Ø,A7,B4,FØ,ØC,85,F9,2Ø,A7,B4,FØ
     ,Ø5,85,F8,4C,EF,BØ,68,68,2Ø, 2749    <Ø88>
1Ø62 DATA 43,B3,4C,5F,B4,2Ø,CF,FF,C9,4C,DØ
     ,Ø9,2Ø,D1,B1,2Ø,48,B2,4C,ØB, 2372    <Ø46>
1Ø63 DATA B6,C9,ØD,6Ø,A9,ØØ,85,5E,2Ø,5F,B4
     ,2Ø,EA,B1,2Ø,ØD,B5,24,5E,3Ø, 2Ø42    <12Ø>
1Ø64 DATA Ø5,2Ø,E4,FF,FØ,FB,2Ø,E1,FF,FØ,26
     ,2Ø,9F,B2,24,5E,1Ø,Ø9,2Ø,4E, 2435    <198>
1Ø65 DATA B5,2Ø,ØD,B5,2Ø,6Ø,B5,2Ø,33,B2,2Ø
     ,3F,B2,9Ø,D7,AØ,B4,A9,28,2Ø, 219Ø    <2Ø7>
1Ø66 DATA FF,B1,2Ø,E4,FF,C9,ØD,DØ,F9,A9,ØØ
     ,85,5E,A5,61,85,FB,A5,62,85, 3Ø56    <24Ø>
1Ø67 DATA FC,2Ø,EØ,B2,4C,64,B1,A5,FC,2Ø,4E
     ,B1,A5,FB,85,FF,2Ø,4E,B1,A9, 3ØØ3    <221>
1Ø68 DATA 2Ø,AØ,3A,2Ø,F2,B1,AØ,ØØ,2Ø,ED,B1
     ,B1,FB,2Ø,4E,B1,C8,CØ,Ø8,9Ø, 2566    <Ø7Ø>
1Ø69 DATA F3,2Ø,ED,B1,24,5E,3Ø,Ø3,A9,12,2C
     ,A9,2Ø,2Ø,D2,FF,2Ø,1Ø,B2,A5, 219Ø    <Ø59>
1Ø7Ø DATA FF,2Ø,4E,B1,A9,92,2Ø,D2,FF,4C,EA
     ,B1,A9,FF,85,B8,85,B9,A9,Ø4, 3Ø73    <Ø29>
1Ø71 DATA 85,BA,2Ø,CØ,FF,A2,FF,4C,C9,FF,2Ø
     ,CC,FF,A9,FF,4C,C3,FF,2Ø,5F, 3315    <189>
1Ø72 DATA B4,A9,8Ø,85,5E,2Ø,4E,B5,2Ø,48,B2
     ,A2,24,A9,2D,2Ø,D2,FF,CA,DØ, 2596    <111>
1Ø73 DATA FA,2Ø,EA,B1,2Ø,EA,B1,2Ø,6Ø,B5,4C
     ,C1,B4,2Ø,B8,B5,A6,5F,A4,6Ø, 2812    <Ø15>
1Ø74 DATA A9,61,2Ø,D8,FF,BØ,ØA,2Ø,B7,FF,29
     ,BF,DØ,Ø3,4C,FB,B4,A9,Ø1,2Ø, 2577    <2Ø1>
1Ø75 DATA C3,FF,2Ø,68,B6,AØ,B4,A9,4F,2Ø,FF
     ,B1,2Ø,F9,B1,4C,FB,B4,2Ø,68, 2921    <237>
1Ø76 DATA B6,A9,37,AØ,B4,2Ø,FF,B1,2Ø,F9,B1
     ,A2,Ø8,C9,44,FØ,Ø6,A2,Ø1,C9, 2717    <213>
1Ø77 DATA 54,DØ,F1,A9,Ø1,A8,2Ø,BA,FF,AØ,ØØ
     ,EØ,Ø1,FØ,1A,A9,4Ø,8D,2Ø,Ø2, 24Ø3    <1Ø1>
1Ø78 DATA A9,3A,8D,21,Ø2,B9,Ø1,Ø2,99,22,Ø2
     ,C8,CC,ØØ,Ø2,9Ø,F4,C8,C8,DØ, 2182    <127>
1Ø79 DATA ØC,B9,Ø1,Ø2,99,2Ø,Ø2,C8,CC,ØØ,Ø2
     ,DØ,F4,98,A2,2Ø,AØ,Ø2,4C,BD, 2Ø18    <Ø25>
1Ø8Ø DATA FF,2Ø,B8,B5,A5,BA,C9,Ø8,9Ø,33,A6
     ,B9,86,57,A9,Ø1,2Ø,C3,FF,A9, 28ØØ    <Ø22>
1Ø81 DATA 6Ø,85,B9,2Ø,CØ,FF,BØ,28,A5,BA,2Ø
     ,B4,FF,A5,B9,2Ø,96,FF,2Ø,A5, 2911    <Ø53>
1Ø82 DATA FF,85,61,A5,9Ø,4A,4A,BØ,13,2Ø,A5
     ,FF,85,62,2Ø,AB,FF,A5,57,85, 2663    <214>
1Ø83 DATA B9,A9,ØØ,2Ø,D5,FF,9Ø,Ø3,4C,A3,B5
     ,86,5F,84,6Ø,A5,BA,C9,Ø1,DØ, 2639    <131>
1Ø84 DATA ØA,AD,3D,Ø3,85,61,AD,3E,Ø3,85,62
     ,4C,FB,B4,A9,13,2Ø,D2,FF,A2, 23ØØ    <12Ø>
1Ø85 DATA 1C,2Ø,ED,B1,CA,DØ,FA,6Ø,ØØ,ØØ,ØØ
     ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ,ØØ, 123Ø    <143>
```

 Listing 2. Der MSE-Lader

Der MSE

Der MSE dient zur Eingabe von Maschinensprache-Programmen. Als erstes müssen Sie den sogenannten »MSE-Lader« (Listing 2) abtippen. Dieser erzeugt erst das eigentliche MSE-Programm auf Diskette oder Kassette.

Wichtig: Vor dem Eintippen des MSE-Laders müssen Sie unbedingt ein paar Befehle eingeben (ohne Basic-Zeilennummer): POKE 44,32 : POKE 8192,0 : NEW

Jetzt können Sie beginnen, das Listing 2 abzutippen. Der MSE-Lader erkennt zwar, wenn Sie beim Eintippen der DATA-Zeilen einen Fehler gemacht haben, aber wenn Sie ganz sicher gehen möchten, sollten Sie den Checksummer vor dem Eintippen aktivieren. Die Prüfsummen für den MSE-Lader finden Sie am Ende der jeweiligen Programmzeilen.

Wenn Sie das Listing 2 nicht auf einmal abtippen möchten, müssen Sie vor jedem neuen Laden des Programms unbedingt die oben genannte POKE-Zeile eingeben!

Wenn Sie alles richtig gemacht haben und das Programm fehlerfrei abgetippt wurde, speichert es sich nach dem Starten selbst auf Diskette oder Kassette unter dem Namen »MSE V1.0«. Dieses fertige MSE-Programm laden Sie dann bei Bedarf wie ein normales Basic-Programm und starten es mit »RUN«.

So arbeitet man mit dem MSE

Als erstes möchte der MSE den Namen des zu bearbeitenden Programms wissen. Dieser steht in der ersten Zeile unserer MSE-Listings. Dann müssen Sie die Start- und Endadresse des Programms eingeben. Dies sind die letzten beiden vierstelligen Hexadezimalzahlen in der ersten Zeile unserer Listings.

Wenn Sie ein Programm von Diskette oder Kassette laden wollen, um an einer bestimmten Stelle weiterzutippen oder noch eine Korrektur vorzunehmen, geben Sie auf die Frage nach der Startadresse ein »L« ein. Danach müssen Sie <D> oder <T> drücken, je nachdem, ob Sie von Diskette oder Kassette (»tape«) laden möchten. Wenn das Programm unter diesem Namen nicht auf der Diskette vorhanden ist oder ein sonstiger Ladefehler vorlag, meldet sich der MSE mit »I/O-ERROR«. In diesem Fall drücken Sie <RUN/STOP RESTORE> und geben einfach noch einmal »RUN« ein.

Beim Abtippen geben Sie nach und nach die abgedruckten Buchstaben und Zahlen des jeweiligen Listings ohne die Freiräume dazwischen ein. Wenn Sie in einer Zeile einen Tippfehler gemacht haben, meldet sich der MSE sofort mit einem Brummton und der Meldung »EINGABE-FEHLER«. Nach einem Druck auf die RETURN-Taste können Sie mit der DEL-Taste den Fehler korrigieren. Wenn Sie das gewünschte Programm vollständig eingegeben haben, speichert es der MSE automatisch auf Diskette oder Kassette.

Bei längeren Listings ist es unwahrscheinlich, daß Sie das komplette Programm auf einmal eingeben. Sie können Ihre bisherige Tipparbeit jederzeit durch <CTRL S> auf Diskette oder Kassette speichern und Ihr Werk später fortsetzen. Sie sollten sich dann allerdings im Heft markieren, wie weit Sie beim Abtippen gekommen sind! Später geben Sie dann nach dem Laden des ersten Programmteils <CTRL N> ein und auf die dann folgende Frage nach der Startadresse die Zeilennummer (Adresse), bei der Sie aufgehört haben zu tippen.

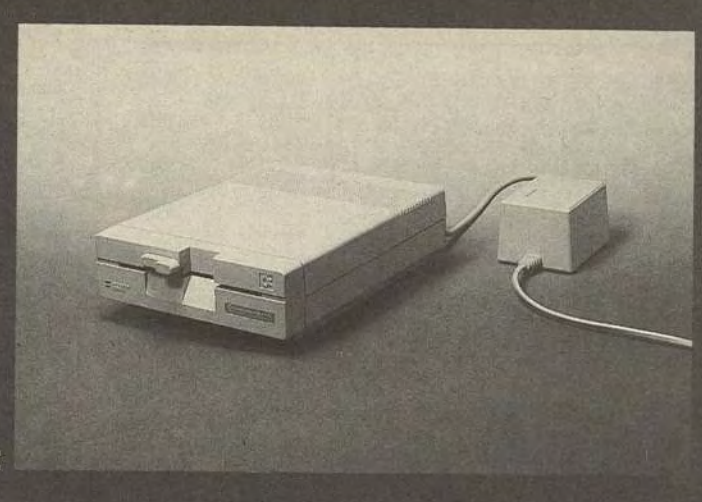
<CTRL M> erlaubt Ihnen jederzeit, Ihr Werk listen zu lassen. Durch <SPACE> können Sie weiterlisten lassen und durch <RUN/STOP> das Listen abbrechen.

Wenn Sie einen Drucker besitzen, können Sie das Programm auch mit <CTRL P> ausdrucken. Mit <CTRL L> wird das Programm noch einmal neu in Ihren C 64 geladen.

(F. Lonczewski/N. Mann/D. Weineck/sk)

SONDERHEFT 41 VORSCHAU

Die Floppy ist das wichtigste Peripheriegerät für den C64. Sie kann weitaus mehr als nur Laden oder Speichern von Programmen. Das Sonderheft 41 informiert Sie umfassend darüber, wie Sie Ihr Laufwerk optimal einsetzen können. Folgende Themen finden Sie im Sonderheft:



■ »Floppy-Kurs«
Lernen Sie die Geheimnisse der Floppy kennen. In diesem ausführlichem Kurs erfahren Sie alles über Funktionsweise, Aufbau und Programmierung der Floppy.

■ »Dateiverwaltung mit der Floppy«
Um große Datenmengen zu verwalten , ist es sehr vorteilhaft, mit »relativen Dateien« zu arbeiten. Dieser Kurs führt Sie Schritt für Schritt in dieses Thema ein.

■ »Anwendungen, Tips und Tricks«
...helfen, die Floppy effektiv einzusetzen. »Disc-Basic« stellt Ihnen 33 neue Befehle für den komfortablen Umgang mit der Floppy zur Verfügung.

Das Sonderheft 41 ist ab dem 28. April erhältlich.

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Hans-Günther Beer
Stellv. Chefredakteur: Gottfried Knechtel – verantwortlich für den redaktionellen Teil
Chef vom Dienst: Susanne Kirmaier
Redaktion: Ralf Sablowski, Elmar Friebe, Klaus Sonnenleiter
Redaktionsassistenz: Brigitte Bobenstetter, Helga Weber (202), Sylvia Derenthal
Hotline: Monika Welzel (640)
Mitarbeiter der Redaktion: Dr. Rudolf Egg

Alle Artikel sind mit dem Kennzeichen des Redakteurs
(kn = Gottfried Knechtel, rs = Ralf Sablowski, ef = Elmar Friebe,
so = Klaus Sonnenleiter
und/oder mit dem Namen des Autors/Mitarbeiters gekennzeichnet

Art-director: Friedemann Porscha
Layout: Erich Schulze (Cheflayouter), Marian Schwarz, Rudolf Weikl, Andrea Miller
Fotografie: Sabine Tennstaedt, Ilona Wiewiorra, Roland Müller
Titelgestaltung: Friedemann Porscha, Rolf Boyke
Spritzgrafik: Ewald Standke
Computergrafik: Werner Nienstedt

Auslandsrepräsentation:
Schweiz: Markt&Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug, Tel. 042-41 5656, Telex: 862329 mut ch
USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063, Telefon: (415) 366-3600, Telex 752-351
Österreich: Markt&Technik Ges. mbH, Hermann Raniger, Große Neugasse 28, A 1040-Wien, Tel. 0043-222-8579455, Telex: 047-132532

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt&Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt&Technik Verlag AG verlegten Publikationen und dazu, daß Markt&Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Produktionsleiter: Klaus Buck (180); Wolfgang Meyer (stellv.) (887)

Anzeigenleitung: Phillip Schiede (399) – verantwortlich für Anzeigen

Anzeigenformate: 1/1 Seite ist 266 Millimeter hoch und 185 Millimeter breit (2 Spalten à 86 Millimeter oder 4 Spalten à 43 Millimeter). Vollformat 297x210 Millimeter.

Anzeigenpreise: Es gilt die Anzeigenpreisliste vom 5. Januar 1988. 1/1-Seite sw: DM 5400,-. Farbzuschlag: erste und zweite Farbe aus der Europa-Skala je DM 1000,-. Vierfarbzuschlag DM 2800,-. Plazierung innerhalb der redaktionellen Beiträge. Mindestgröße 1/4-Seite.

Anzeigenverwaltung und Disposition: Lisa Landthaler (233)

Anzeigen-Auslandsvertretung: England: F. A. Smyth&Associates Limited, 23a, Aylmer Parade, London, N2 OPQ. Telefon:0044/1/3405058, Telefax: 0044/1/3419602
Taiwan: Third Wave Publishing Corp., 1-4 Fl. 977 Min Shen E. Road, Taipei 10581, Taiwan, R.O.C., Tel. 00886/2/7630052, Telefax: 00886/2/7658767, Telex: 078529335

Vertriebsleiter: Helmut Grünfeldt (189)

Verkaufsleiter Abonnement: Benno Gaab (740)

Verkaufsleiter Einzelhandel: Robert Riesinger (364)

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1,

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 4613-249. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 14,–

Druck: SOV Graphische Betriebe, Laubanger 23, 8600 Bamberg

Sonderdruck-Dienst: Alle in dieser Ausgabe erschienenen Beiträge sind in Form von Sonderdrucken zu erhalten. Anfragen an Reinhard Jarczok, Tel. 089/4613-185, Fax 4613-776.

Redaktionsdirektor: Michael M. Pauly

Vorstand: Otmar Weber (Vors.), Bernd Balzer

Leiter Unternehmensbereich »Populäre Computerzeitschriften«:
Eduard Heilmayr, Werner Pest

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: Markt&Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0, Telex 5-22052

ISSN 0931–8933

Telefon-Durchwahl im Verlag: Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089/4613 und dann die Nummer, die in den Klammern hinter dem jeweiligen Namen angegeben ist.

Mitteilung gemäß Bayerischem Pressegesetz:
Aktionäre, die mehr als 25% des Kapitals halten: Otmar Weber, Ingenieur, München; Carl-Franz von Quadt, Betriebswirt, München; Aufsichtsrat: Carl-Franz von Quadt (Vorsitzender), Dr. Robert Dissmann (stellv. Vorsitzender), Eduard Heilmayr




GEOS
POWER PACK

